JANUARY 2003 - Volume II - Issue 1

php | architect

The Magazine For PHP Professionals

Writing Secure PHP Code

Make your applications safer

Reviewed for you:

IonCube PHP Accelerator 1.3.3
CodeCharge Studio 1.0

Plus:

Using the .NET Assembly with PHP
Writing A Web-based PDF Viewer
Taming Full-Text Search with MySQL
Accessing the Windows API and Other DLLs
Implementing Database Persistence Layers

This copy is registered to:
Jakob Breivik Grimstveit
jakob.grimstveit@starshipping.com

The designers of PHP offer you the full spectrum of PHP solutions

>Develop
>Protect
>Scale

Serve More. With Less.

Zend Performance Suite

Reliable Performance Management for PHP

Visit **www.zend.com**
for evaluation version and ROI calculator

Zend
Technologies Ltd.

php | architect

INDEX

Departments

Features

EDITORIAL RANTS

There is nothing like a "trial by fire" to make or break your day. By the time the first issue of php|a hit the virtual stands, we had worked insane hours for almost a straight month to ensure that everything was as good as we could possibly make it. Even though, as a result, we were terribly tired, I couldn't sleep for two days (yes, I have been committed to-and released from-a mental institution since then, in case you were wondering).

Luckily, the results were encouraging-better than we had originally expected, to be sure,and the December issue did very well. Many of you wrote us to let us know that the magazine was better than you had expected in terms of content and detail; my personal favorite was a note found in a web forum, where someone had written that they were surprised by the amount of information contained in php|a, as he was expecting a ten page fanzine. Still, we had a few constructive critiques sent our way, and that was very good-it gave us guidance on where we had been less than brilliant and, therefore, a range of issues to fix before the next month.

As a result, you will find a number of changes in this issue. First of all, we have now included internal links throughout the magazine, for those who like to read it on-screen. The table of contents on page 3 is fully linked to each of the articles, while the words "php|architect" at the bottom of each page link back to the table of contents. This should make "flipping" through the pages of the magazine easier for everyone.

With this issue, I think we have improved the quality of the topics we cover as well. We wish our role in the PHP community to be that of helping to make our beloved language an invaluable choice for enterprise-level projects, and that can only be done by increasing the quality of the information available out there. Whether we succeed or not is, as they say, for posterity to decide, but at least we're trying!

The other good news, as you no doubt will have already noticed by the time you read this, is that this issue is free. It's our way to say thank you to all those who have believed in us-and welcome to those who are just now getting a chance to try us out.

Finally, I'm happy to say that, from this issue forward, Brian K. Jones

joins our editorial team. After the December issue was out, we thought we had done a pretty good job, but it became evident that there should be at least one person in our editorial staff for whom English is the first language. Brian brings his excellent technical expertise and valuable knowledge of the editorial process to our team, and we can only be happy to have him with us (note to Brian-you now officially owe me a drink. Make that a good one).

Happy reading!

php|architect

Volume II - Issue 1
January, 2003

Publisher

Marco Tabini

Editors

Arbi Arzoumani
Brian K. Jones
Marco Tabini

Graphics & Layout

Arbi Arzoumani

Administration

Emanuela Corso

Authors

Arbi Arzoumani, Shawn Bedard, John W. Holmes, Jayesh Jain, David Jorm, Theo Spears, Marco Tabini, Leon Vismer

php|architect (ISSN 1705-1142) is published twelve times a year by Marco Tabini & Associates, Inc., P.O. Box. 3342, Markham, ON L3R 6G6, Canada.

Although all possible care has been placed in assuring the accuracy of the contents of this magazine, including all associated source code, listings and figures, the publisher assumes no responsibilities with regards of use of the information contained herein or in all associated material.

Contact Information:

General mailbox:	info@phparch.com
Editorial:	editors@phparch.com
Subscriptions:	subs@phparch.com
Sales & advertising:	sales@phparch.com
Technical support:	support@phparch.com

Copyright © 2002-2003 Marco Tabini & Associates, Inc. — All Rights Reserved

PHP 4.3 Is Out

The new version of PHP is out. RC4 was the last step before the final release—developers have been asked to commit to the PHP CVS repository only changes that fix bugs marked as "critical" in the bug tracking system, and thankfully only lasted a few days.

The PHP developers have also solved a dilemma that has all but dominated the mailing lists of late—the naming of the CLI (command-line interface) version of PHP vs. the CGI executable, which is used when using PHP as a separate executable to run scripts through a web server.

Artware Releases New PHP-based CMS

Vienna, Austria-based Artware Multimedia announced in early December the publication of its new Content Management System based on PHP, called Constructioner Web Technology.

Conceived to be an inexpensive content management solution aimed primarily at non-technical users, Constructioner features a web-based WYSIWYG management system, support for an arbitrary number of languages, and can be integrated with Dreamweaver. The software is available for free download for developers—there are no time limits on the trial version. A key, which retails for $399 (US), must be obtained only when the CMS engine is used in a public website.

A live sample is available at http://www.constructioner.com, where you can play with the product and also find more information about it.

ExpertRating Launches PHP Certification Exam

Online certification company ExpertRating has launched a new certification exam for PHP developers.

The exam takes place entirely online and consists of forty multiple-choice questions, each feature between two and eight different answers, one or more of which could be correct—your basic mid-term nightmare. The questions cover topics ranging from the basics of the PHP language—like operators, syntax, and typecasting—to more advanced subjects like regular expressions, sessions and mailing.

The ExpertRating exam costs $10 (US), which can be paid online through a secure credit card form. According to the ExpertRating website, the exam must be taken within a month of registering for it.

For more information, you can follow this link: http://www.expertrating.com/details.asp?examid=91.

php|a

We Dare You To Be A Professional.

Subscribe to php|a Today and Win a book from Wrox Press

php|architect

The Magazine For PHP Professionals

Reviewed For You

CodeCharge Studio 1.0

By Arbi Arzoumani

The Cost:
CodeCharge: $149
CodeCharge Studio: $279.95

Requirements:
Windows '95/'98/ME/NT4/2000/XP
64MB RAM
20MB Hard Drive Space
File Size 16.7MB

Download Page:
CodeCharge

Download limitations:
The CodeCharge download is a fully functioning 30-day trial

CodeCharge Home Page:
CodeCharge

Company Background:
YesSoftware Inc. develops and markets RAD (Rapid Application Development) tools. They are based in Delaware and are a self-funded and privately held company. They begin developing the code generation technology in 1999. It was completed in 2001.

Let me start by confessing to the fact that this was *not* an easy review. To fully appreciate this product, you must sit down and actually use it for a practical purpose. It's a great learning experience. In a nutshell, CodeCharge is a powerful code generating software for the web. Some of you might think "Oh no, another code generator—run for the hills!". However, I would like to put CodeCharge in its own category, rather than just call it a code generator. To fully explore this application, I would have to include at least 30 screen shots of its intuitive interface to configure, manage, modify, and publish your project. I suggest that, once you read this review, go to their website and download the trial copy and start using it.

Let's start by covering the basic grounds. The installation was straightforward. If you don't have a license, you can use the program for 30 days. It supports code generation for the following programming languages: ASP.NET (C#), ASP 3.0, PHP 4.0, Java Servlets 2.2, JSP 1.1, ColdFusion 4.01, and Perl 5. How? It uses an XSL engine using XML file formats. If you think this is another Visual Studio, think again—this baby can generate code and let you manage it with ease.

CodeCharge comes with a list of code generating wizards called 'Component Builders'. Some of the 'Component Builders' are:

Grid Builder - Lets you quickly create database grids on your pages. This is great for those back end management tools.

Record Builder - Rapidly create data maintenance forms. This is handy for both front-end and back-end pages (ie. registrations forms).

Login Builder - What's a back-end without a proper security login page. Or, have your users login to their accounts.

On top of these code generating wizards, sits the Application Template Library, a set of built-in templates for common web solutions that can be configured and launched in no time. The version of CodeCharge that I had came with the following solutions: Employee Directory, Bug Tracking, Task Manager, Portal, Registration Form, Forum, and a Book Store.

Naturally when creating a project, you are asked if you want one of the above solutions or a blank project. A blank project is not actually 'blank'—there are at least 6 common files that are always included in the source of every page. These common files can be regarded as the framework behind the IDE.

For those of you using MS FrontPage, here's the good news. You can get an add-in and convert your MS FrontPage into a powerful code-generating monster. Take a look at Figure 1 for the added CodeCharge toolbars.

CodeCharge comes with a complete IDE interface. Before publishing any of the pages generated with the Application Builder, the developer can modify anything from the HTML to the code. There are 5 different views of a certain page:

Design - A WYSIWYG editor for manipulating your page. You can drag and drop different components right into your page. For example, an input box, or a submit button.

HTML - Here you can edit the HTML directly. Since the HTML code is kept separate from the actual server side code, it's easier to modify any visual elements of a page.

Code - As expected, this is a fully syntax highlighted editor. It is less colorful compared to the built-in PHP show_source() function, but it does the job.

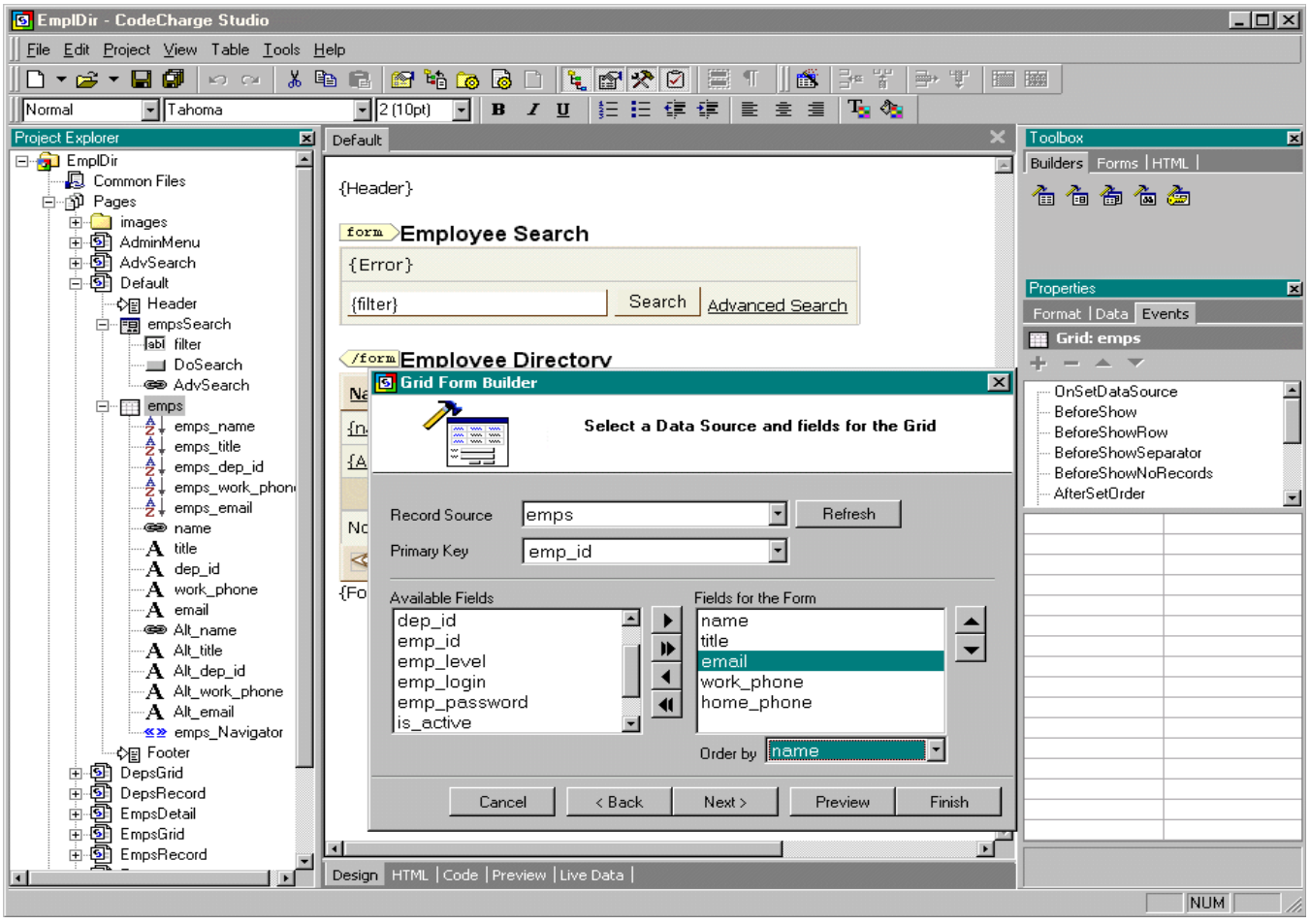
Preview - This mode will display the HTML code without any special tags (seen in the design mode).

Live Page - By providing the URL to the live site, the page is displayed live from the server.

It's possible to define multiple database connections in the same project. This can be useful to pull data from different sources—for example, user data stored in a MySQL server, and product catalogue data stored in a different location on Oracle. The code generator includes support for the following database libraries: JET, ODBC, JDBC, ADO, DBI, and PHPLib.

The database connection properties was a little confusing to setup. I had some trouble setting up a connection string to a MySQL server. (the product kept on asking me for a ODBC driver).

One of great features of CodeCharge lies in its flexibility: the user can modify any generated code prior to publication. All modifications are locked and are not



overwritten during any subsequent code generation.

Some of you are probably wondering how good is this generated code actually is. Lets not forget that it comes from templates written by other developers—other than the common files discussed earlier, the rest of the code is fairly well commented. It can be easily understood and modified by an intermediate developer—as long as that person understands the language that the code was generated in. The code language can be changed at anytime during the development process. Here's a tip for all of you generating PHP code: Make sure you define your session.save_path in your php.ini file prior to previewing your code on the live server. The PHP code uses session_start() in its common files. Another thing I noticed is that any modifications made to the common files will be overwritten—I guess you can't change the framework.

What CodeCharge does not provide is a debugging tool. However, the folks at YesSoftware have come up with some nifty XSL templates to generate code. In the near future, users will be able to generate their own templates, themes, components and even wizards using an SDKthat is currently in the works.

Also, version 2.0 of CodeCharge will be released shortly, and some of the new features being planned include:

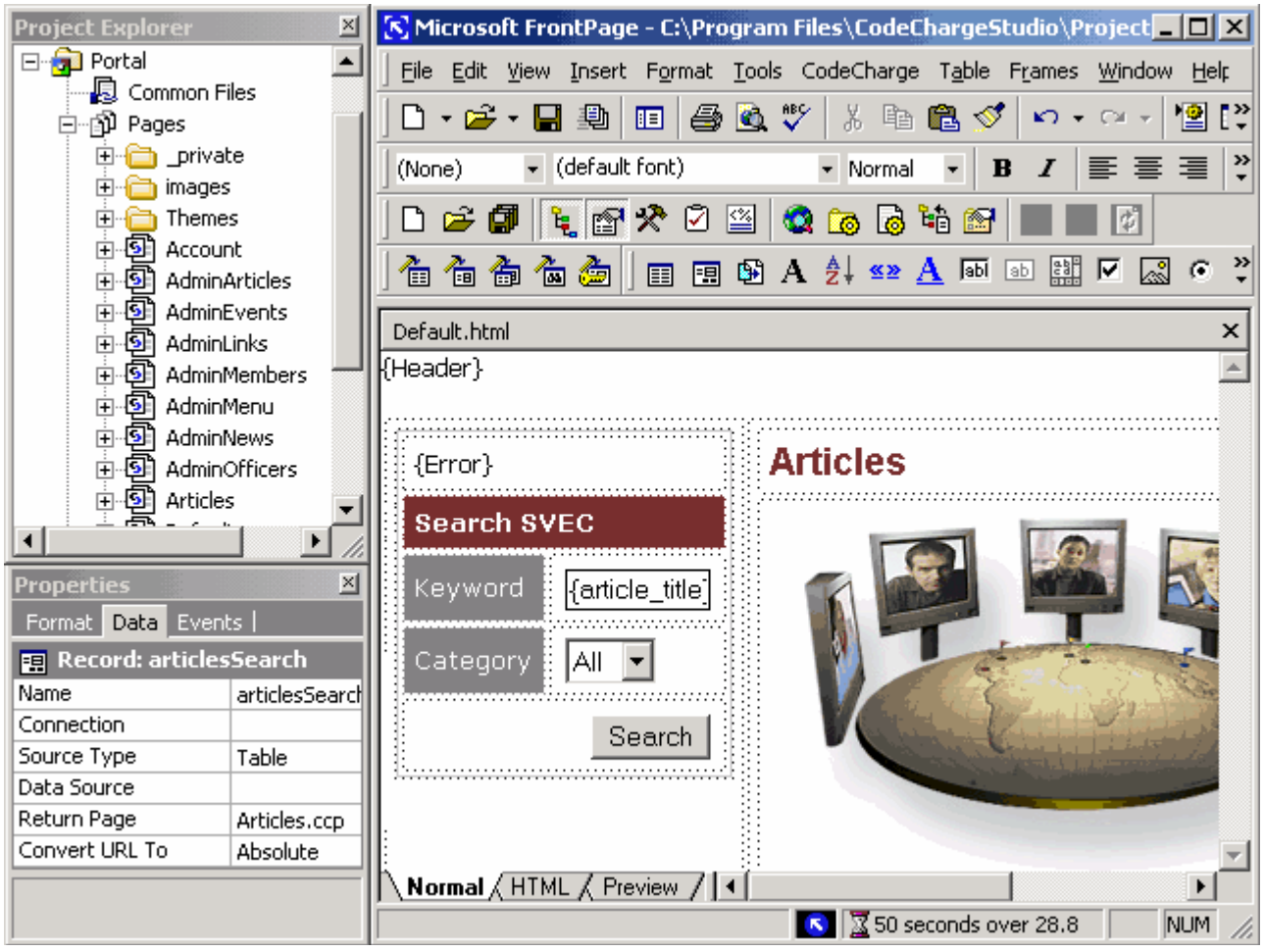
-Integration with source control and versioning systems
-VB.NET support
-NT and LDAP authentication
-Generation of site diagrams and test scripts
-Additional components and builders
-Enterprise CRM, CMS and team collaboration solutions built with CodeCharge Studio

Conclusion

As I suggested before, anyone interested in this application should download the trial version and check it out—there are a lot of other features that I did not have time to cover in this review. From what I heard version 2.0 is going to be a big upgrade. For its price tag this code generation application is well worth it. One great application that I can see this being used for is creating prototypes of web applications in very short periods of time. In other words, last minute proposals.

php|a

Figure 1



e-xact transactions

Payment Processing Technology

MOVING BUSINESS. REAL TIME. REAL SMART

They don't get IT!

- Banks do what banks do.
- Take your money.
- Use your money.
- Charge you to access your money.

We get IT!

- We build software.
- Our code works.
- Download via the Web.
- Bank Agnostic.
- 24 x 7 Test System

You get IT!

- **FREE**
- http:developer.e-xact.com
- Influence our developement

E-xact Transactions (Canada)
Suite 304 - 134 Abbot Street
Vancouver, BC, V6B 2K4
Canada

Contact Information
Toll Free: 1-877-303-9228
Ph: 604.691.1670
Fax: 604.691.1678

Web: www.e-xact.com
Email: comments@exact.com

Trading Symbol: EXZ.U
Listing: (CDNX)

FEATURES

Implementing Database Persistence Layers in PHP

By Shawn Bedard
Jig Technologies

The OOP functionality built into PHP makes it possible to access information stored in a database by using a structured, portable and easily extensible approach--not to mention that you can get rid of those ugly SQL statements embedded in your scripts!

Introduction

When PHP was first developed by Rasmus Lerdorf in 1995, it was a cute little scripting language for form processing and personal web pages. Everything you needed was right there in handy global variables. Although this allowed for rapid development of form processing scripts, it scaled very poorly due to the lack of scope control, data abstraction and extensibility. Inexperienced developers loved it because it was quick and easy, but the seasoned software engineers cringed in fear and agony as its popularity gained. There was no real attempt to support an organized Object Oriented (OO) structure. When I was first introduced to PHP, I was a little uncomfortable with its architecture. It was more like an object spaghetti structure if you tried to build anything of substance.

The focus went from scripting up pages to building real software applications.. Developers were actually starting to build sizable web applications in PHP and it was not going well. Fortunately, the creator of PHP was helped out by Andi Gutmans and Zeev Suraski to move the language to a new stage. It became more than just a scripting environment. It became a viable development platform. The release of version 4 in May 2000 was enhanced for better encapsulation and a reasonably sound OO programming environment. But, being a first release, it was buggy, a bit slower than it needed to be and it had a lot of other rough edges. Since then, there has been a lot of good work optimizing and extending the language beyond its initial buggy and awkward structure. This has allowed for more traditional OO frameworks to be created for PHP. This article describes the development of one of those frameworks--the persistence layer.

With the increased use of object classes in PHP, the need to persist these objects on a permanent basis becomes apparent. What's the use of objects if you can't keep them around for a while? Unfortunately, most common relational databases do not make it easy to stuff an object in them, nor is it reasonable to store your objects in the session as sessions tend to disappear. However, the development of a persistence layer framework in PHP addresses this object persistence problem. The persistence layer is a collection of classes that allows developers to store and access objects or classes from a permanent source. In theory this source can be a file or memory space but in practice data is

REQUIREMENTS

PHP Version: **4.0 and above**
O/S: **Any**
Additional Software: **N/A**

generally stored in a database.

First I will explain the concepts of OO programming and how persistence layers are used for applications built in that manner. With that foundation, I will outline three approaches to designing persistence layers. Finally, I will describe the architecture of persistence layers using the robust persistence layer approach and how it can be applied in common application development.

Object Oriented Programming and Persistence Layers

As previously mentioned, a persistence layer is used for persisting objects in a permanent fashion. This concept is not a new one--persistence layers have been very prevalent in more "developed" languages like Java, C++ and even newer enterprise frameworks like .NET. However, we have only very recent developments to thank for PHP's emergence as a real OO architecture, allowing for constructs such as persistence layers to become a viable addition to an application.

Although some developers are very familiar with objects and OO programming, it is worth discussing just what this is, in order to provide a bit of context. OO programming is a type of programming in which programmers define data structures to group related pieces of data. This data is accessed and acted upon by accessor functions. In this way, the data structure becomes an object where internal data is encapsulated and accessed through these functions. The resulting created objects can have relationships in a hierarchical fashion known as inheritance. Thus, objects can inherit attributes and functionality from other objects by an inheritance mechanism. OO programming is the implementation of this object relationship.

When developing an OO-architected system, it is often very useful to develop an object model describing what objects exist in a system. This object model will also describe how these objects relate to each other. The object modeling exercise allows architects to view the system in a graphical format before any code is created. This helps to ensure that the application being built can be extended, maintained and effectively developed. At the end of the day, what you end up with is a class (business object) for each of the objects created in the object model.

One of the principal advantages of OO programming techniques over their older procedural programming ancestors is that they enable programmers to create modules that do not need to be changed when a new type of object is added. A programmer can simply create a new object that inherits many of its features from existing objects. This makes object-oriented programs easier to modify.

The other advantage of OO programming is the ability to pass structured objects instead of long arrays or miscellaneous groups of variables such as really long parameter lists in a method call or 32 session variables. All this is eliminated by architecting a good object model. The long parameter list gets replaced by a single object. The 32 variables in the session can be stored in 1 or 2 nicely formed objects.

There are, however, some not-so-pleasant qualities of OO programming in PHP that should be noted. One of these qualities is the performance hit in creating the object constructs and calls to access data. However, the performance hit is no longer a real concern for most applications as memory prices decrease and processor speeds increase. So long as your app can reside in memory and ultra high performance is not a concern, this performance hit will be negligible. Another quality is the complexities of storing objects in common permanent storage mechanisms such as databases and file systems. This complexity can be abstracted away from the developer in the form of a persistence layer. A persistence layer will enable the developer to effectively and easily store business classes into a permanent storage.

In PHP, it might not always be appropriate to use this kind of OO programming style. For instance, if you are simply creating a form with a few fields and passing on the data in an email, the overhead of an OO framework is not likely necessary. No data is saved, and no state really needs to be preserved. Since a persistence layer is in the business of persisting objects or classes, it does not make sense to be using it if you don't have any objects to persist. However, if that form feeds data into an application that needs to access the data at a later time, then OO development makes a whole lot of sense.

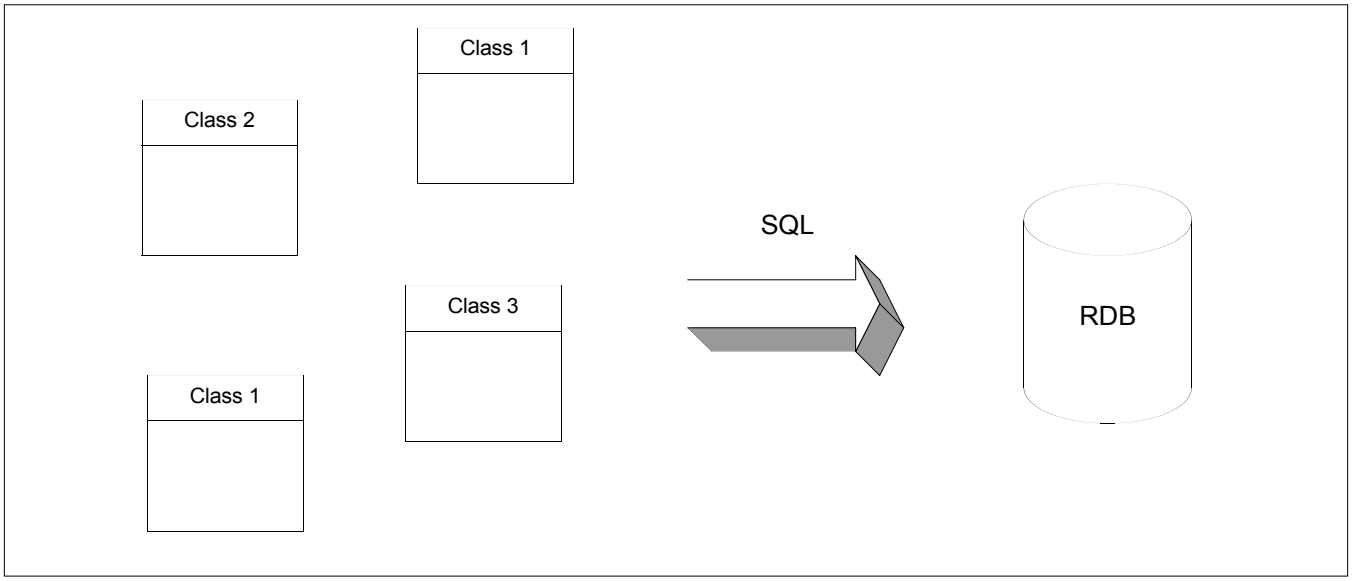
Approaches to Persistence Layers

Not all persistence layers are created equal. There are various approaches to creating this kind of framework. Some require more effort and are good for larger applications and some are much lighter weight and are appropriate for smaller applications.

Hardcoding SQL

The most common and least palatable approach to developing a persistence layer is hardcoding Structured Query Language (SQL) in the business classes. (See Figure 1) Many people develop this type of persistence layer without even knowing it. To illustrate, lets take the example of creating a User class or object with the attributes of name, uid, and pwd. The User class would need the SQL "select name, uid, pwd from…" embedded in the class to retrieve the necessary data from the database and a routine to bind the results to the appropriate local member variables of the class. Likewise, an insert, update and delete would require the same kind

Figure 1 - Hard-coding SQL in your domain/business classes.



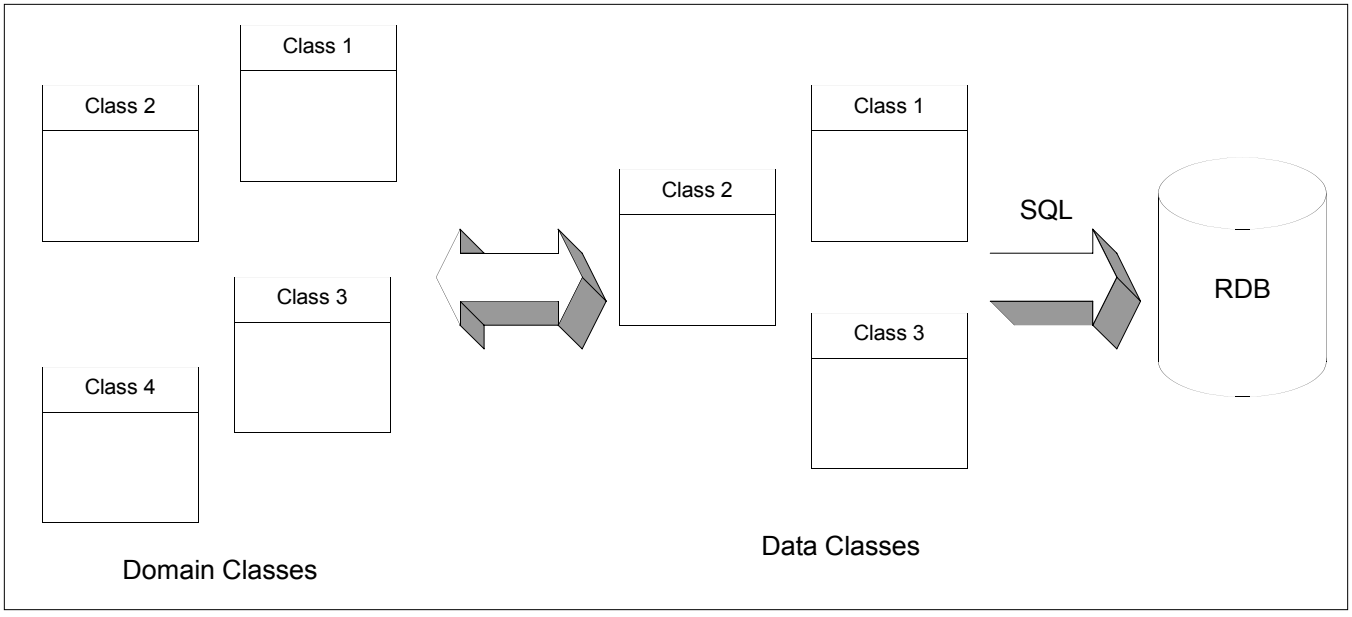
of routine and SQL.

The advantage of this approach is that it allows you to write code very quickly and it is a viable approach for small applications and prototypes. The disadvantage is that it tightly couples business classes with the schema of your relational database. This would mean that a simple change such as renaming a column may require a reworking of the code. In the above User example, adding a column would require at least six changes to your code. The insert, update and select would need a change in the SQL plus each of the local variable update routines. This would be multiplied by the number of domain classes that accessed that table. One can quickly see how maintenance mayhem can develop with larger systems.

Data Classes

A slightly better approach to developing persistence layers is where the SQL statements for your business classes are encapsulated in one or more "data classes." (See Figure 2) Once again, this approach is suitable for prototypes and small systems of less than 40 to 50 business classes. This helps abstract things a bit more because one or more domain classes can access a single data class. For instance, you could have the domain classes 'user login' and 'user update' accessing a single user object. User login would simply call the user object's 'select' method with the userid and password set. If a row is found then the login object returns true,

Figure 2 - Hard-coding SQL in your separate date classes.



otherwise a false is returned. The user update would simply set all the object values and run the update method.

The major disadvantage here is that SQL and object update routines are still hardcoded. This means that adding or removing a column needs to have three or more SQL statements changed along with the object updating routines. However, if modeled correctly, this kind of update will only have to be done once per table change. This was not the case in our previous example. Previously, this kind of change had to be done for each domain class that accessed this table. So, for those who need to access the same data in 32 different ways need to change 32 domain classes! Since most folks are happier to update one object instead of 32, this new approach makes for a more joyous PHP developer when DB changes are made. After all, no one likes a grumpy PHP developer!

Other examples of this approach include developing stored procedures in the database to represent objects (replacing the data classes of Figure 2) and Microsoft's ActiveX Data Object (ADO) "strategy". The best thing that can be said about this approach is that you have at least encapsulated the source code that handles the hard-coded interactions in one place: the data classes. This approach also helps keep SQL out of the domain classes, making them more readable.

Unfortunately, not all database systems support stored procedures-this is particularly true of many open-source servers like MySQL-and therefore this approach may not be suitable for your purposes.

Robust Persistence Layer

A robust persistence layer maps objects to persistence mechanisms in such a manner that simple changes to the relational schema do not affect your object-oriented code. (See Figure 3) If we use the example above of adding or removing a column in a table we can see how our job becomes somewhat easier. If you are adding a new column to a particular object you do not have to change the three SQL statements and an update routine in the code. Instead all that is needed is to update the one data object to include the extra local member and the persistence layer takes care of the rest. All of a sudden, your once-frustrated PHP developer has started dancing for joy at all the extra time he has saved in making DB updates. This means more time to attend rave parties--or whatever it is that interests PHP developers these days.

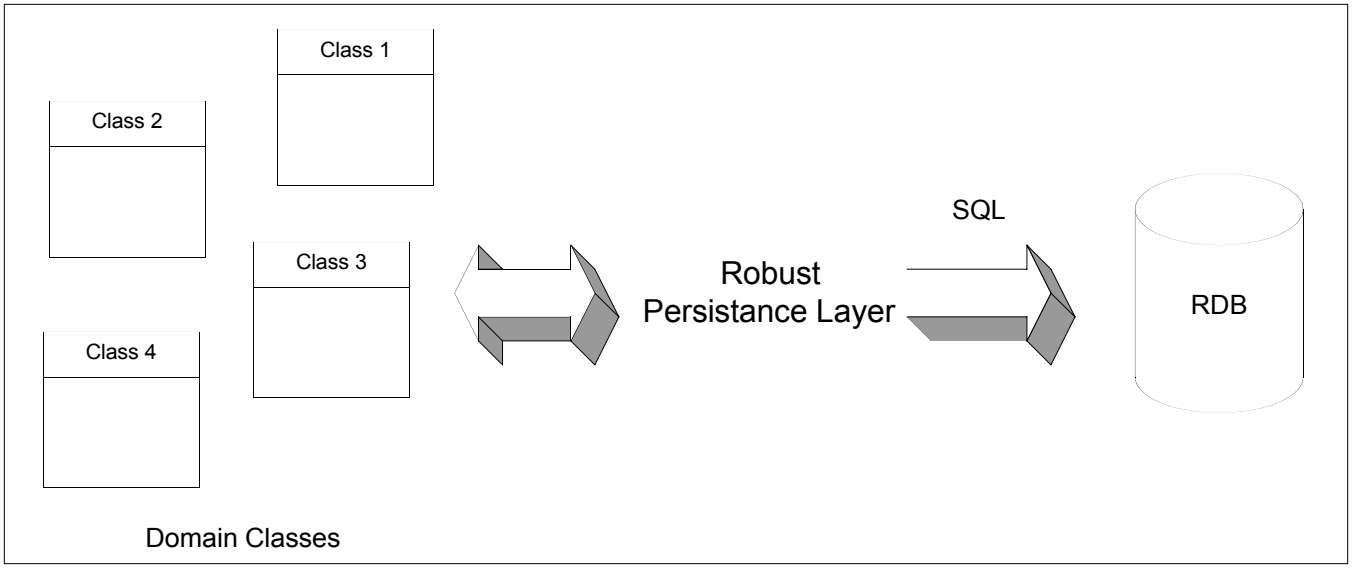
The main advantage of this approach is that your application programmers do not need to know a thing about the schema of the relational database. In fact, they do not even need to know that their objects are being stored in a relational database. This approach allows your organization to develop large-scale, mission critical applications. However, there is a performance impact to your applications--a minor one if you build the layer well, but there is still an impact.

Example Persistence Layers

Now that several approaches have been outlined at a high level, we will present details of a particular implementation. This implementation aims to provide the robust persistence layer described above. It was developed to separate the database layer from the business layer giving three distinct advantages. First, removing the hardcoded SQL makes the application easier to extend. Second, complex applications are more maintainable because a lot of the data access code is centralized in a single location. Finally, code in the business layer becomes cleaner and more readable.

In practice, it is difficult to build an efficient persist-

Figure 3 - Robust data persistence layer.



ence layer that will provide the developer with all the mechanisms available in SQL directly . As such, this implementation described does not support the complete array of relational DB functionality abstractly. Support for DB features like grouping, sorting and internal DB functions can be added to this framework but this entails extra processing which in turn will lead to a greater performance hit. Also, the extra code needed would cause the framework to use more memory. As a result, this support has not been added into the framework explicitly.

To avoid the use of "special case" SQL by a PHP developer within the framework, we can opt to write SQL in the business classes. If a true OO programming practice was being adhered to this would not have to be done. However, this is a compromise between abstracting the DB away from the business class developer 100% and developing a framework that is efficient and lightweight.

There are basically three parts to this framework: configuration, logging and the persistence layer itself. The configuration drives all the settings for this framework. It includes things like db connection info and logging parameters. The logging system is an extremely simplified logging mechanism that outputs informational messages to a text file specified in the configuration. The actual persistence layer is a set of objects to be extended and code that dynamically generates the SQL to persist object information via the database.

One of the principal advantages of OO programming techniques over their older procedural programming ancestors is that they enable programmers to create modules

Architecture Details

The actual implementation of the theory is likely to be of greatest interest to developers looking to use a persistence layer. Below in figure 4 is the class diagram for an implementation of a robust persistence layer. The most interesting classes are those classes that can be extended to create persistent data objects. Those objects are DBObject, DBSetObject, and DBMultiObject.

DBObject basically represents a single row in the database or a single instance of an object. As such, it has the ability to select, update, insert, and delete. All these functions can be used multiple times using a single instance of the object. This means that during the lifetime of this object multiple updates can take place without recreating the object several times. This is important for performance in situations where an insert and many updates need to take place.

DBMultiObject represents a single row of two or more joined tables in the database. Due to the nature of this query it is a read only object and can only perform selects. Generally this kind of limitation is not a problem as this kind of access is typically read only anyway. However, if a write is needed, the ids can be obtained from the resulting select to create the appropriate DBObjects. Since at the end of the day this is really a join between two tables, a join clause must be specified to join the tables appropriately. This unfortunate necessity makes this object less of an abstraction and more complicated for the end developer. However, it provides flexibility and possible performance gains over two separate queries.

The DBSetObject object is a utility class for using in set data. Although by itself this represents a single row in the database result, it can be used in conjunction with DBSet or DBMultiSet to provide a representation for multiple rows. DBSet is used to query multiple rows in a single table and DBMultiSet is used to query multiple rows in a multiple table join. Due to the latter, DBSetObject is a read only object and supports select operations only. Ideally, it would be able to support the same insert, update and delete as DBObject, but that would be beyond the scope of this article.

Implementation

Now that you have an idea of how this possible architecture might work, there's nothing like some good old fashioned functional implementation examples to clarify things. These examples will illustrate how this implementation can be used. Let's start with a couple of tables. For illustration purposes we have chosen user and address where one address can have many users

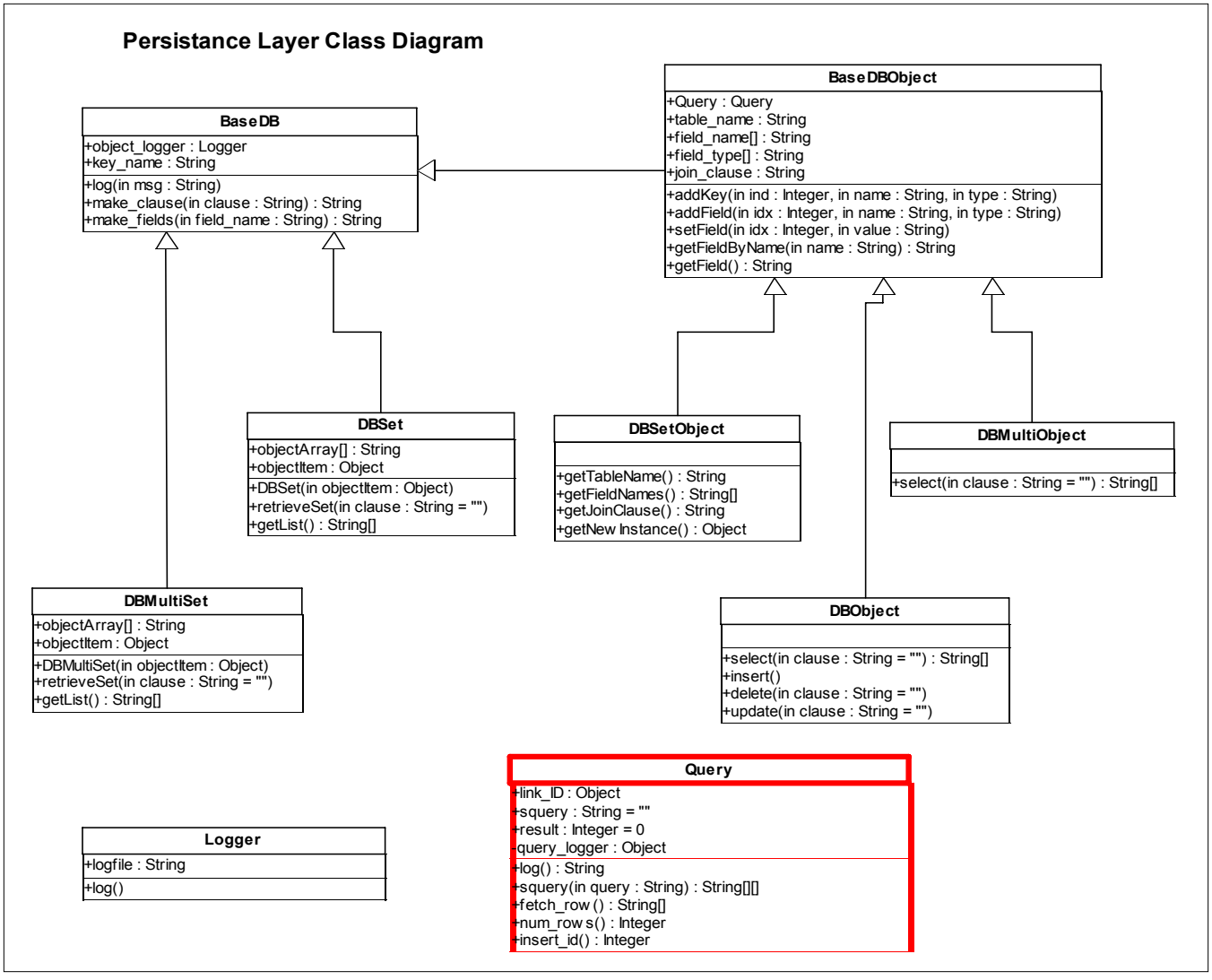
Figure 5

```
CREATE TABLE users (
  user_id INT PRIMARY KEY
              auto_increment,
  password varchar(15) default NULL,
  first_name varchar(63) default NULL,
  last_name varchar(63) default NULL,
  client_id varchar(63) default NULL,
  address_id INT(2)
) TYPE=MyISAM;

CREATE TABLE address(
  address_id INT(2),
  street     VARCHAR(30),
  city       VARCHAR(30)
) TYPE=MyISAM;
```

Listing 1 - user.php

```php
<?php

// Define all the columns as global statics
define ( "USER_UID", 0 );
define ( "USER_PWD", 1 );
define ( "USER_FNAME", 2 );
define ( "USER_LNAME", 3 );
define ( "USER_CID", 4 );

class Users extends DBObject {
 function Users () {
  // Call parent constructor
  $this->DBObject();
  // Set the keys and fields
  $this->log ( "Initializing Object as a Users" );
  $this->addKey (  USER_UID,  "user_id",     DB_STRING );
  $this->addField (  USER_PWD,  "password",     DB_STRING );
  $this->addField (  USER_FNAME,  "first_name",     DB_STRING );
  $this->addField (  USER_LNAME,  "last_name",     DB_STRING );
  $this->addField (  USER_CID,  "client_id",     DB_STRING );
  // Defing the table to be operated on.
  $this->table_name = "Users";
 }
}

?>
```

Figure 4 - Class diagram of example implementation



Listing 2

```php
<?php

require "include/pldb/All.php"; // include persistence framework
require "include/db_objs/user.php"; //include user object (above)

// create the user object, set the primary key to 1, and select it to
// obtain the data in the object.
$user = new Users();
$user->setField(USER_UID,1);
$user->select();

// print out some info.
echo $user->getField(USER_FNAME)." "$user->getField(USER_LNAME);
echo "has the userid ". $user->getField(USER_UID)

// update a field and update save it to the database.
$user->setField(USER_UID,"newuserid");
$user->update();

// print out some info and notice the change.
echo $user->getField(USER_FNAME)." "$user->getField(USER_LNAME);
echo "has the userid ". $user->getField(USER_UID)

// delete that object from the database
$user->delete();

?>
```

(see Figure 5).

The easiest thing one might want to do is obtain and manipulate a single user. As mentioned above, DBObject gives the developer the ability to insert, update, select and delete. If these methods are called with no parameters, these actions will be performed based upon the value of the key(s). However, the user has the option to overwrite this functionality by passing in the 'where' clause. Passing in the clause breaks the abstraction of this layer but gives the developer the ability to handle special cases. Listing 1 is an example of extended DBObject as the file user.php.

Using the object is relatively straight forward. A select, update and delete looks like Listing 2.

As you can see here, we have done a select, update, and delete with relatively few lines of code and no SQL.

Listing 3 - An example of the UserList extending DBSet in the file userlist.php

```php
<?php

// Define all the columns as global statics
define ( "USERLIST_UID", 0 );
define ( "USERLIST_PWD", 1 );
define ( "USERLIST_FNAME", 2 );
define ( "USERLIST_LNAME", 3 );
define ( "USERLIST_CID", 4 );

class UserList extends DBSetObject {
 function UserList () {
  // Call parent constructor
  $this->DBSetObject();

  $this->log ( "Initializing Object as a UserList" );

  $this->addKey (  USERLIST_UID,  "user_id",     DB_STRING );
  $this->addField (  USERLIST_PWD,  "password",     DB_STRING );
  $this->addField (  USERLIST_FNAME,  "first_name",     DB_STRING );
  $this->addField (  USERLIST_LNAME,  "last_name",     DB_STRING );
  $this->addField (  USERLIST_CID,  "client_id",     DB_STRING );
  $this->table_name = "Users";

 }
     function getNewInstance() {
        return new UserList();
     }
}

?>
```

Listing 4 - Accessing data

```
<?php

require "include/pldb/All.php"; // include persistence framework
require "include/db_objs/userlist.php"; //include user object (above)

// create the user object, set the primary key to 1, and select it to
// obtain the data in the object.
$userlist = new DBSet(new UserList());
$userlist->retrieveSet( "where user_id > 0" );
$ulArray = $userlist->getList();


// print out some info.
foreach($ulArray as $key => $valueObj)  {
    echo "User's ID:".$valueObj->getField(USER_UID)."<br>";
    echo "User's First Name:".$valueObj->getField(USER_FNAME)."<br>";
}

?>
```

The business class developer has a very readable clean set of code. In addition, developing further functionality using this object is very quick and easy.

Those of you who are familiar with this type of work know that it is not always practical to retrieve one object at a time. It is often necessary to use a collection of objects. This is often used for listing out information in the system. The DBSet and DBSetObject objects are used for this type of listing. For the moment these collections are read only so only the select feature can be used. Listing 3 is an example of the UserList extending DBSet in the file userlist.php.

Like DBObject, the select statement can be passed a 'where' clause to specify which rows to retrieve. However, if no clause is specified then this call will simply retrieve all the rows in the table specified. Using the object is a bit more complicated because the results are returned in an object array. So accessing the data must be done in an iterative fashion like the for loop illustrated in Listing 4.

Although this example has SQL, it is not necessary if the desire is to retrieve all the rows from the table. Again, most of the details about how the data is being retrieved have been abstracted away from the developer making the code simpler and more readable. If the developer wanted to retrieve another list from the table, no new objects would need to be created, as the userlist object could be reused again.

It is often desirable to obtain data from two or more different tables during the same transaction. In some cases it is possible to retrieve the required data with one access to the database. For performance reasons one query is usually preferred over two or more. This implementation has taken this possibility into account, in addition to queries across multiple tables. The DBSetObject and the DBMultiObject allow users to specify multiple tables in the form of a join clause. An example of the DBSetObject being extended is shown

Listing 5 - An example of the DBSetObject being extended

```
<?php

define ( "TU","users" );
define ( "TA ","address" );
class UserMultiList extends DBSetObject {
 function UserMultiList () {
      // Call parent constructor
      $this->DBSetObject();

       $this->log ( "Initializing Object as a UserMultiList" );

       $this->addKey (  USERLIST_UID, TU.".user_id", DB_STRING );
       $this->addField (  USERLIST_PWD, TU.".password", DB_STRING );
       $this->addField (  USERLIST_FNAME, TU.".first_name", DB_STRING );
       $this->addField (  USERLIST_LNAME, TU.".last_name", DB_STRING );
       $this->addField (  USERLIST_CID, TU.".client_id", DB_STRING );

       $this->addField (  USERLIST_CITY, TA.".city", DB_STRING );
      $this->addField (  USERLIST_STREET, TA.".street", DB_STRING );
$this->join_clause = TU." LEFT JOIN ".TA." ON ".TU.".
                     user_id=".TA.".address_id";
 }

?>
```

Listing 6

```
1  <?php
2
3 $userMultiList = new DBMultiSet(
                        new UserMultiList());
4 $userMultiList->retrieveSet( "" );
5 $ulMultiArray = $userMultiList->getList();
6
7 foreach($ulMultiArray as $key => $valueObj)  {
8       echo $valueObj->getField(USER_UID);
9       echo $valueObj->getField(USER_STREET);
10 }
11
12 ?>
```

in Listing 5.

Like DBSet the retrieveSet method can be passed a where clause to specify which rows to retrieve. However, if no clause is specified then this call will simply retrieve all the rows in the table join specified. As before, accessing data must be done in an iterative fashion like the for loop illustrated in Listing 6.

As you can see, there is no SQL code in this listing. All the details of the data layer have been abstracted away from the developer.

Concluding Remarks

Hopefully, this article provides some insight as to why a persistence layer is useful as well as some insights to various implementations. This kind of implementation is not at all useful for a 'build-a-form-in-fifteen-minutes' kind of project. However, if you find yourself building larger applications with a complex set of logic in the backend, this abstraction layer is invaluable in trying to keep your application maintainable.

As shown in the example above, you can easily persist objects using a database with little or no SQL and very little PHP code. This allows the developer to create applications that are maintainable and extendable. Furthermore, faster development will be facilitated by abstracting away the complexities of persisting an object via a database. These advantages are best achieved using the third approach. Using frameworks like this will allow developers to build PHP applications that can quickly adapt to the changing requirements of software.

Acknowledgments

I would like to thank Raymond Gigliotti for helping produce some of the PHP code for the persistence layer example. Also, I would like to thank Nancy Lam for helping to better formulate my ideas for this article.

php|a

Shawn Bedard is a senior architect based in Toronto, Canada. The code he presents in this article is based on his database persistence layer code available at https://sourceforge.net/projects/dbpl. You can reach Shawn at sabedard@jig.to.

It feels better when they let you touch it.

Anyone can teach PHP. But getting your hands on the keyboard is the quickest and surest way for you to learn everything from PHP and MySQL to advanced topics like classes, objects, templates, and PDFs.

That's what we do at TAPInternet. You walk in. Sit down. Grab a keyboard. You walk out feeling confident that you can handle anything the PHP/MySQL world can throw at you.

Five days of Pure Pleasure.

Tap Internet provides everything you need: Hands-on building of dynamic web sites in a lab designed specifically for teaching. Eight hours of daily instruction for five full days. Over 40 hours of hands-on topics ranging from basic PHP to output buffering and template techniques.

Did we Neglect to Mention Fun?

Everything is designed to keep you motivated and learning, including 'Quick Challenges' that give you a really good feel for how to use the language. It keeps things interesting. Heck, it keeps *you* interesting, because you're interacting with other students as you learn.

Scott Nichols from Shoe Carnival told us that what he liked best was, "...the personal attention and ability to ask questions ... I was treated as a guest."

Not exactly what you'd expect from a geek, eh?

Log on Now to Learn More.

Want to master PHP/MySQL in a heartbeat? Then beat it on over to http://www.tapinternet.com/php and see how much more there is to TAPInternet.

Or give any of the guys a call: 1-866-745-3660.

But do it now. The next course is about to begin.

TAP INTERNET
PHP COURSES

Hands-on training from the get-go.

Co-sponsored by Zend Technologies

Classes enrolling now. 1-866-745-3660

http://www.tapinternet.com/php/

FEATURES

Accessing the Windows API and other Dynamic Link Libraries

By David Jorm

All intrinsic and most third party libraries for Windows are DLLs (Dynamic Link Libraries). DLLs are a compiled set of functions or objects, similar conceptually to UNIX's .so (Shared Object) files. This article shows you how you can take advantage of them from within PHP.

The Windows API provides a collection of functions spread across several DLLs allowing a programmer to perform any Windows task: constructing and manipulating windows and graphical widgets, accessing the file system, accessing networking configuration and reading and writing the registry and event log. Many of these features are not (yet) provided for by PHP extensions, but a new experimental extension, w32api (Windows 32 API) allows PHP programmers to call the functions and member objects of any Windows DLL. This extends both the features of the Windows platform and the myriad of libraries built on it to PHP programmers.

Using the w32api Extension

The w32api extension contains only 5 functions, one of which is defunct as of PHP 4.2.3:

```
bool w32api_deftype (
                     String TypeName,
                     String MemberType,
                     String MemberName)
```

Used to define a data type for use with other w32api functions. Any number of MemberType and MemberName arguments may be passed, one for each member variable of the type. For those unfamiliar with data types, they are a collection of variables of differing types, accepted as arguments of some Windows API functions.

```
resource w32api_init_dtype(
              String TypeName, Mixed Value)
```

Used to create an instance of a data type defined with `w32api_deftype()`. Any number of Value arguments may be passed, one for each member variable of the type.

```
bool w32api_register_function       (
              String Library,
              String FunctionName,
              String ReturnType)
```

Used to register functions from the Windows API or other windows DLLs. The Library argument should be the name of the DLL containing the function. The path to the DLL is not required as this is provided by the

REQUIREMENTS

PHP Version: **4.0 and above**
O/S: **Windows**
Additional Software: **N/A**

DLL's registry entry. Most Windows API functions have two names, one of which is an alias. When giving the FunctionName argument, the alias must be used. For example, you would use "MessageBeepA" instead of "MessageBeep". The ReturnType argument must be either string, long, int or bool.

```
mixed w32api_invoke_function (
                    String FunctionName,
                    String FuncArgument)
```

Defunct as of PHP 4.2.3, used to invoke a function registered with `w32api_register_function()`. Any number of FuncArgument arguments may be passed, one for each argument accepted by the Windows API function being called. Functions registered with `w32api_register_function()` are now available as normal PHP functions, called by the name with which they were registered, so this function is no longer required.

```
void w32api_set_call_method (Int Method)
```

Sets the call method used by the w32api extension. The Method argument must be one of the constants `DC_CALL_CDECL` or `DC_CALL_STD`. `DC_CALL_STD` is the default.

To explore these functions, we will cover three examples:

1. A site logging security violations to the event log
3. A page displaying windows system information
2. Enhancements to a command line application using the Windows API

Accessing the Event Log

The Windows platform contains an integrated logging and log management framework called the Event Log. The Event Log provides a standard, centralized mechanism for all applications to record software and hardware events. When an error occurs, the system administrator must determine what caused the error, attempt to recover any lost data and prevent the error from recurring. It is helpful to this process if both the operating system and userland applications record significant events. The system administrator can use the event log to help determine what conditions caused the error and the context in which it occurred.

Event Log Design

The Event Log is comprised of five key components:

Three Log Files: Application, System and Security The Event Log data is stored in seperate flat log files for each category of event. It is possible to add extra log files representing different categories of events, but this is not normally required.

The Event Log Viewer: The Event Log Viewer is an administrative tool that ships with Windows NT, 2000 and XP that is used to view, filter and manage the contents of the event log files.

w32api (Windows 32 API) allows PHP programmers to call the functions and member objects of any Windows DLL

Windows API Functions: The Windows API exposes functions from advapi32.dll providing the programmatic ability to read and write to and from the event log and to register and unregister event sources.

Binary Format Message Files: The messages given by each event source are defined using ASCII text in a binary format message file. This is a DLL built by the Visual Studio MC (Message Compiler), which automatically assigns numbers to each message and generates a table of the message text. The event logging functions provided by the Windows API use the messages defined in this file to write the description of each event to the event log. Message files allow for text in various languages to be set as the description of each event. The event logging service will write the description to the event log in the language appropriate to the operating system's native language.

Event Source Registry Entries: Each application or service writing to the event log should have its own event source. Event sources are defined in the registry by creating a subkey underneath:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Event Log\{Log Name}\{Event Source Name}
```

This subkey must have two values:

EventMessageFile: A REG_SZ value of the absolute path (environment variables allowed) to the message file DLL containing the event descriptions for this event source.

TypesSupported: A REG_DWORD value representing which event types are supported by

this event source. This is a bitmask created by ORing one or more of the values shown in Figure 1. For example, Error, Warning and Information types have the value 0x0007.

Figure 1

Event Type	Value
Error	0x0001
Warning	0x0002
Information	0x0004
Audit Success	0x0008
Audit Failure	0x0010

Building a Message File

The first step to building a message file is to create an .mc file defining the message resource table. This is simply an ASCII text file with an entry for each event in the format shown in Figure 2.

For our example web application, the .mc file should be as shown in Figure 3.

Compiling this message file into a resource-only DLL (the most efficient kind) is then a three step process, assuming the .mc file is named messages.mc:

1. Use the Message Compiler to create an .rc file from the .mc file using the command:

```
mc messages.mc
```

Figure 2 - .mc file format and description

Text Entry	Purpose
# Comment Here	Comments are preceded by #
MessageID=0x1	Hexadecimal value of the Message ID
Severity=Warning	Severity of this event. Can be Error, Warning, Success or Information
SymbolicName=WWW_AUTH_FAILURE	Name of the event.
Language=English	Language of the message description.
A user has failed to authenticate for the web application.	Description of the event.
.	Single period to terminate the message.

Figure 3 - message.mc file

```
MessageId=0x1
Severity=Warning
SymbolicName=WWW_AUTH_FAILURE
Language=English
A user has failed to authenticate for the web application.
.
MessageId=0x2
Severity=Success
SymbolicName=WWW_AUTH_SUCCESS
Language=English
A user has successfully authenticated for the web application.
.
MessageId=0x3
Severity=Error
SymbolicName=WWW_AUTH_NOFILE
Language=English
The web application was unable to open the authentication file.
.
```

This should create two files:

```
MSG00001.bin
messages.rc
```

2. Use the Resource Compiler to create a .res file from the .rc created by the Message Compiler:

```
rc -r -fo messages.res messages.rc
```

3. Use the linker to create a .dll file:

```
link -dll -noentry -out:messages.dll messages.res
```

Using these options will create a resource-only DLL. These are smaller and faster than normal DLLs. Finally, messages.dll should be copied to `c:\WINNT\System32\`.

Creating an Event Source

The next step is to link the message file DLL we have created to the name of the event source. This is done by creating a single key in the windows registry. Since the web application will want to log to the 'Application' log with a source name of 'WebApp' this key will be:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Event Log\Application\WebApp
```

The key will require values:

```
Name: EventMessageFile
Type: REG_SZ
Data: %SystemRoot%\System32\messages.dll

Name: TypesSupported
Type: REG_DWORD
Data: 7
```

%SystemRoot% will be substituted by the system's root directory, by default `c:\WINNT`. This key can be added manually using regedit or by creating a .reg file and importing it into regedit. Neither of these tasks are within the scope of this article.

Reporting Events

Reporting an event is a three step process, involving registering an event source, reporting an event from that event source and, finally, deregistering the event source. These tasks are handled by three functions exposed by advapi32.dll:

```
long RegisterEventSourceA (String ServerName,
                           String SourceName)

long ReportEventA (Long LogHandle,
                   Long Type,
                   Long Category,
                   Long EventID,
                   Mixed UserID,
                   Long NumStrings,
                   Long DataSize,
                   Long Strings,
                   Mixed RawData)

long DeRegisterEventSource ( Long LogHandle )
```

The `LogHandle` argument of `ReportEventA()` must be a handle returned by `RegisterEventSourceA()`. The `NumStrings`, `DataSize`, `Strings` and `RawData` arguments are all used to provide strings of additional data with an event report. This data is shown in the Event Viewer as either bytes or DWORDs. This article only covers events without strings of additional data attached.

Putting It All Into a Class

Since event logging is a clumsy process to replicate each time an event is reporting, we can simplify matters by creating and event logging class, as shown in Listing 1.

Building The Authentication Page

The final step is to construct a HTML form accepting a user's authentication credentials so we can authenticate them and report the triggered events to the event log. This is done in Listing 2.

Console Enhancements

Although PHP contains intrinsic functions allowing for the display and collection of data on a command line console, the Windows API can enhance this functionality for Windows console applications. API calls including message boxes and sounds can be used to create PHP console applications with the same interface and features of native windows console applications.

Console Title

Setting and retrieving the title of a command line console is one of the easiest tasks within the Windows API. Many functions require, as we have seen earlier, user defined types and constants to work, neither of which are required for this task. The Windows API exposes two functions from kernel32.dll used to set and retrive a console title:

```
bool SetConsoleTitleA (String Title)

long GetConsoleTitleA (String TitleBUffer,
                       Long BufferLength)
```

There are two important points to note with these functions. The first point is that the first argument taken

Listing 1

```
<?php

class eventlog {

    // Properties
    // Build an array of the log type constants and their corresponding values
    var $log_types=array(
                        array("EVENTLOG_SUCCESS", 0),array("EVENTLOG_ERROR_TYPE", 1),
                        array("EVENTLOG_WARNING_TYPE", 2), array("EVENTLOG_INFORMATION_TYPE", 4),
                         array("EVENTLOG_AUDIT_SUCCESS", 8), array("EVENTLOG_AUDIT_FAILURE", 10)
                         );
     var $logtype;
     var $hEventLog;

     // Constructor
     function eventlog() {

         // Register w32api functions
         w32api_register_function("advapi32.dll", "RegisterEventSourceA", "long");
         w32api_register_function("advapi32.dll", "ReportEventA", "long");
         w32api_register_function("advapi32.dll", "DeregisterEventSource", "long");

         // Register event source
         $this->hEventLog=RegisterEventSourceA(NULL, "WebApp");
     }

     // Report Event Method

     function reportevent($eventid, $logtype) {
         $logtypefound=false;

         // Match the string log type passed to a value from the array of constants
         for ($i=0; $i<sizeof($this->log_types); $i++) {
             if ($this->log_types[$i][0] == $logtype) {
                 $this->logtype=$this->log_types[$i][1];
                 $logtypefound=true;
             }
         }
         if (!$logtypefound)
             return false;

       // Report the event
       if (!ReportEventA($this->hEventLog, $this->logtype, 0, $eventid, NULL, 0, 0, NULL, NULL))
             return false;
       return true;
     }

     // Destructor
     function destructor() {

         // De register the event source
         DeregisterEventSource($this->hEventLog);
     }
}
?>
```

by **GetConsoleTitleA()** is a buffer to contain the string returned by the function, while the function's return code is an indicator of success. When providing a buffer using a language with explicitly typed variables such as C or Visual Basic, you must fill the buffer with space characters in order for it to be long enough to hold the required data. AlthoughPHP uses scalar variables, the buffers must still be spaced out in orderto work with the Windows API. This can be done easily using **str_repeat()**. The following line of code would fill **$TitleBuffer** with 255 space characters:

```
$TitleBuffer = str_repeat(" ", 255);
```

Secondly, when using **SetConsoleTitleA()**, the console title will only remain set while the program has control of the console. Once the program exits and control returns to cmd.exe, the console title reverts to that set by cmd.exe.

Sounds

Basic waveform sounds can be played using the **MessageBeep()** function from user32.dll. This func-

Listing 2

```
<html>
<head>
    <title>Authentication Form</title>
</head>
<body>
<!— Begin dynamic content —!>
<?php
$u='testuser';
$p='testpass';
// Instantiate an eventlog object

if (isset($u) && isset($p)) {
    if (auth($u, $p)) {
        echo "<h2>Authentication Successfull!</h2>\n</body></html>\n";

        // Instatiate an eventlog object, report the event and destroy the object
        $el_obj = New eventlog();
        $el_obj->reportevent(2, "EVENTLOG_INFORMATION_TYPE");
        $el_obj->destructor();
        unset($el_obj);
        exit;
    } else {
        echo "<font color=\"#FF0000\"><h2>Authentication Failed</h2></font>\n";

        // Instatiate an eventlog object, report the event and destroy the object
        $el_obj = New eventlog();
        $el_obj->reportevent(1, "EVENTLOG_WARNING_TYPE");
        $el_obj->destructor();
        unset($el_obj);
    }
}

function auth($u, $p) {
    if ($u == 'testuser' && $p == 'testpass')
        return true;
    return false;
}

class eventlog {

    // Properties
    // Build an array of the log type constants and their corresponding values
    var $log_types=array(
        array("EVENTLOG_SUCCESS", 0), array("EVENTLOG_ERROR_TYPE", 1),
        array("EVENTLOG_WARNING_TYPE", 2), array("EVENTLOG_INFORMATION_TYPE", 4),
        array("EVENTLOG_AUDIT_SUCCESS", 8), array("EVENTLOG_AUDIT_FAILURE", 10));
    var $logtype;
    var $hEventLog;


    // Constructor
    function eventlog() {

        // Register w32api functions
        w32api_register_function("advapi32.dll", "RegisterEventSourceA", "long");
        w32api_register_function("advapi32.dll", "ReportEventA", "long");
        w32api_register_function("advapi32.dll", "DeregisterEventSource", "long");

        // Register event source
        $this->hEventLog=RegisterEventSourceA(NULL, "WebApp");
    }

    // Report Event Method
    function reportevent($eventid, $logtype) {
        $logtypefound=false;

        // Match the string log type passed to a value from the array of constants
        for ($i=0; $i<sizeof($this->log_types); $i++) {
            if ($this->log_types[$i][0] == $logtype) {
                $this->logtype=$this->log_types[$i][1];
                $logtypefound=true;
```

Continued On Page 25...

Listing 2: Continued From Page 24...

```
72              }
73          }
74          if (!$logtypefound)
75              return false;
76
77          // Report the event
78          if (!ReportEventA($this->hEventLog, $this->logtype, 0, $eventid, NULL, 0, 0, NULL, NULL))
                return false;
79          return true;
80      }
81
82      // Destructor
83      function destructor() {
84          // De register the event source
85          DeregisterEventSource($this->hEventLog);
86      }
87  }
88
89  ?>
90  <!— End dynamic content —!>
91  <form action="eventlog.php" method="post">
92  <table border="0" cellspacing="3" cellpadding="3">
93  <tr>
94  <td><b>Username:</b></td>
95  <td><input type="text" name="u"></td>
96  </tr>
97  <tr>
98  <td><b>Password:</b></td>
99  <td><input type="text" name="p"></td>
100 </tr>
101 <tr>
102 <td colspan="2"><input type="submit" value="Authenticate"></td>
103 </tr>
104 </table>
105 </form>
106 </body>
107 </html>
```

tion is designed for use when notifying the user of an error, so only a few sounds are available. These sounds are defined by constants passed as the only argument to the function:

```
bool MessageBeep(Unsigned Integer SoundType)
```

The possible values of SoundType are shown in Figure 5.

The wave file played for each sound is defined by an entry in the registry. This can be edited easily using the 'Sound Events' section of the 'Sounds and Multimedia' control panel.

Message Boxes

Although constructing windows and positioning widgets within Windows is a complex process, simple windows such as message boxes and common dialog boxes can be constructed and displayed in just one Windows API call. Message boxes are particularly usefull for displaying errors or usage information for a command line program. They can be created using the `MessageBoxA()` function from user32.dll:

```
long MessageBoxA (   Long WindowHandle,
                     String Text,
                     String Caption,
                     Long WindowType)
```

The WindowHandle argument can be null (use the intrinsic PHP constant NULL). The WindowType argument must be one of the constants listed in Figure 5.

A Simple Integrity Tool

Our sample command line application to which we can apply these enhancements is an integrity tool checking that a file's md5 checksum matches that of a known good state. It's shown in Listing 3.

Applying the Enhancements

This console application can be enhanced using the three features of the Windows API we just examined. The application can set the console title while it is running, emit a sound when a checksum mismatch is found and display errors in a message box instead of on the console. Listing 4 shows our enhanced integrity tool.

Figure 5

Constant	Value	Meaning
	-1	Simple Beep
MB_OK	0	Default Sound
MB_OCONASTERISK	64	Asterisk Sound
MB_ICONEXCLAMATION	48	Exlamation Sound
MB_ICONHAND	16	Hand Sound
MB_ICONQUESTION	32	Question Sound

Windows System Information

One of the key uses of the Windows API is to set and retrieve system information such as configuration information, performance counters and operating system build numbers. In this example we will build a HTML page generated by PHP displaying several elements of system information, including some that must be extracted from a user defined data type.

System Information Functions

Several functions for viewing system information are exposed by kernel32.dll, some allowing a single piece of information to be retrieved, others filling a type with several elements of data. Just a few of these functions will be used in this example:

```
void GetSystemInfo (SYSTEM_INFO SysInfo)

long GetSystemDirectoryA (
            String Buffer, Long BufferSize)

long GetVersion()
```

Note that the **GetSystemDirectoryA()** function accepts a buffer as an argument. This must be spaced out using **str_repeat()** before being passed to the function, as discussed in example 2. Two important things should be noted about the **GetSystemInfo()** function. Firstly, it is a subroutine rather than a func-

Listing 3

```
1  <?php
2
3 // Define our files and their known good checksums
4
5 $files=array(
6             array('C:\WINNT\explorer.exe', '6fd321ccbd0eeb6189c714443b215c64'),
7             array('C:\WINNT\php.ini', 'd21410157a5a20242e408d048a301c37')
8             );
9
10 // Loop through each file, notifying if it differs from the known good state
11
12 for ($i=0; $i<sizeof($files); $i++) {
13
14     // If the file does not exist, display an error and exit
15
16     if (!file_exists($files[$i][0])) {
17         echo('Unable to hash file: ' . $files[$i][0]);
18         sleep(1);
19         exit;
20     }
21     if (md5_file($files[$i][0]) != $files[$i][1]) {
22         echo "Checksum mismatch for file: " . $files[$i][0] . "\n";
23     }
24 }
25
26 ?>
```

Listing 4

```php
<?php

// Define Windows API Constants
define("MB_ICONEXCLAMATION", 48);

// Define our files and their known good checksums
$files=array(
        array('C:\WINNT\explorer.exe','6fd321ccbd0eeb6189c714443b215c64'),
        array('C:\WINNT\php.ini', 'd21410157a5a20242e408d048a301c37')
             );

// Register Windows API functions
w32api_register_function("kernel32.dll", "SetConsoleTitleA", "bool");
w32api_register_function("user32.dll", "MessageBeep", "bool");
w32api_register_function("user32.dll", "MessageBoxA", "long");

// Set the console title
SetConsoleTitleA("PHP Integrity Tool");

// Loop through each file, notifying if it differs from the known good state
for ($i=0; $i<sizeof($files); $i++) {

    // If the file does not exist, display an error and exit
    if (!file_exists($files[$i][0])) {

        // Display this error in a message box
        MessageBoxA(NULL, 'Unable to hash file: ' . $files[$i][0],
        "PHP Integrity Tool", MB_ICONEXCLAMATION);
        sleep(1);
        exit;
    }
    if (md5_file($files[$i][0]) != $files[$i][1]) {
        echo "Checksum mismatch for file: " . $files[$i][0] . "\n";

        // Emit an Exclamation sound
        MessageBeep(MB_ICONEXCLAMATION);
    }
}

?>
```

tion, so essentially returns void, although this is not a return type accepted by **w32api_register_function()**. Instead, the type 'bool' can be used. Secondly, **SYSTEM_INFO** is a user defined data type containing several long integer elements. It is defined using **w32api_deftype()** as shown in Figure 6:

Although w32_api functions can be used to define and enumerate types, there is no capacity to retrieve a type's individual elements.

User Defined Data Types

The **w32api_deftype()** and **w32api_init_dtype()** functions return/refer to user defined data types as a dynaparm PHP resource. PHP resources are user defined data types stored internally in the Zend engine whose member elements are retrieved using abstraction functions within the Zend engine. Since there are no PHP functions to extract the elements of a dynaparm resource, the only way to extract a type's element is to write a PHP extensional-lowing this or address the type as a PHP scalar variable and manually unpack its contents. The latter is by far the easiest. A type is simply a binary string assembled by concatenating the binary strings of its member elements. For example, an enumerated **SYSTEM_INFO** type appears as a binary string which appears in hexdecimal (run through **bin2hex()**) as:

```
000000000010000000
000100fffffe7f0100
0000010000004a0200
000000010006000608
```

Figure 6

```
w32api_deftype("SYSTEM_INFO", "long", "dwOemID", "long", "dwPageSize",
               "long", "lpMinimumApplicationAddress", "long",
               "lpMaximumApplicationAddress", "long", "dwActiveProcessorMask", "long",
               "dwNumberOfProcessors", "long", "dwProcessorType", "long",
               "dwAllocationGranularity", "long", "dwReserved");
```

This string can be unpacked into its member elements using the PHP `unpack()` function:

```
array unpack (String Format, String Data)
```

The `Format` argument is a format string defining how the data should be broken up, with each element defined by the format string being returned as a named element of an array. Format strings are a complex topic not within the scope of this article, but in simple terms the string is comprised of a data type, element name and delimiter character for each element. For example, a long element named mylong and a string element named mystring would be extracted with the format string "lmylong/smystring".

A Page Displaying System Information

Once problems such as buffer handling and type element retrieval have been overcome, assembling a page that displays system information is a relatively easy process. Since the name of each element of the `SYSTEM_INFO` type must be used twice, once to generate the format string and once to extract the named elements of the array returned by `unpack()`, we will build an array of the elements which can be looped through in each case. Listing 5 shows a small PHP script that performs these tasks.

Conclusion

The Win32 API interface provided by PHP makes a great number of powerful functions available to your scripts. Although taking advantage of this functionality will limit the portability of your code, depending on the type of application you are writing it might well be worth it!

php|a

David works as a document imaging and OCR programmer for a small Australian company. He spends his spare time writing PHP code and studying environmental science.

Listing 5

```
<!— HTML before dynamic content —!>
<html>
<head>
    <title>Windows System Information</title>
</head>
<body>
<h1>Windows System Information</h1>
<table border="1">
<th>Property</th>
<th>Value</th>
<!— Begin dynamic content —!>
<?php

// Since we are using a scalar to hold the value of the SYSTEM_INFO type, no
// types need to be defined or enumerated.

// Register Windows API functions

w32api_register_function("kernel32.dll", "GetSystemInfo", "bool");
w32api_register_function("kernel32.dll", "GetSystemDirectoryA", "long");
w32api_register_function("kernel32.dll", "GetVersion", "long");

// Build an array of all elements in a SYSTEM_INFO type

$type_elements=array("dwOemID", "dwPageSize", "lpMinimumApplicationAddress",
                     "lpMaximumApplicationAddress", "dwActiveProcessorMask",
                     "dwNumberOfProcessors", "dwProcessorType",
                     "dwAllocationGranularity", "dwReserved");

// Space out the buffer to allow it to hold the contents of a SYSTEM_INFO type

$buf=str_repeat(' ', 255);

GetSystemInfo($buf);

// Loop through each element of the type, building a format string

$first_element=true;
```

Continued On Page 29...

Listing 5: Continued From Page 28...

```
39 for ($i=0; $i<sizeof($type_elements); $i++) {
40     if (!$first_element)
41         $unpack_str .= "/";
42     else
43         $unpack_str="";
44     $unpack_str .= "l" . $type_elements[$i];
45     $first_element=false;
46 }
47
48 // Unpack the SYSTEM_INFO type
49 $arr=unpack($unpack_str, $buf);
50
51 // Loop through each element of the type, displaying its value in a HTML table
52
53 for ($i=0; $i<sizeof($type_elements); $i++) {
54     echo "<tr><td>" . $type_elements[$i] . "</td><td>" .
55 $arr[$type_elements[$i]] . "</td></tr>\n";
56 }
57
58 // Space out a buffer again for use with GetSystemDirectoryA()
59
60 $buf=str_repeat(' ', 64);
61
62 // Run our other Windows API functions and display their results in a HTML table
63 GetSystemDirectoryA($buf, strlen($buf));
64
65 echo "<td>SystemDirectory</td><td>" . $buf . "</td></tr>\n";
66
67 echo "<tr><td>Version</td><td>" . GetVersion() . "</td></tr>\n";
68
69 ?>
70 <!— End dynamic content —!>
71 </table>
72 </body>
73 </html>
```

FEATURES

Taming Full-Text Search with MySQL

By Leon Vismer

How manytimes has your boss asked the dreaded question, 'Can we add a full text search engine to the clients' news articles?'.

In the past, developers tried to accomplish this with a tool like htdig or some home-grown indexing library written in C or Perl. In the end, they could only do so much with regular expressions and very creative SQL 'like' queries. Some of us went as far as creating 'bad words' tables with a dictionary, mapping these into a lookup table, with an initial parser to break up sentences to make it all work.

MySQL 4.0 to the rescue

The development and beta release of MySQL 4.0 brings some welcome text indexing features to the open source community. Although the 3.23 version does allow for full-text indexing, version 4.0 added some mature enhancements like moving configuration options to the config files and allowing fully featured boolean search functionality.

How to get started

Before we continue, I presume that MySQL 4.0 has been installed on your servers. Figure 1 shows the creation of a new database and the creation of the article table with some article examples. Save the listing as database.SQL and the following command will create the database and tables needed

```
$ MySQL -uadminuser -padminpassword < database.SQL
```

with `adminuser` and `adminpassword` being the username and password you use for administrating your MySQL databases.

Creating The Full-Text Index

Looking at our table article, we would like to be able to search on author and the story itself. First connect to the articles database and the alter command will create the full-text index:

```
MySQL> alter table article add FULLTEXT search_idx (author, story);
```

You can also create a full-text index at the time of table creation by adding FULLTEXT (author,story) to the create table command.

REQUIREMENTS

PHP Version: **4.0 and above**
O/S: **Any**
Additional Software: **MySQL 4.0 and Above**

Figure 1

```
drop database if exists articles;
create database articles;
use articles;
create table article (
       id int auto_increment not null,
       author char(64) not null,
       story text not null,
       PRIMARY KEY (id)
);
insert into article values(0, 'Leon Vismer', 'PHP (recursive acronym for \"PHP: Hypertext
Preprocessor\") is a widely-used Open Source general-purpose scripting language that is especially
suited for Web development and can be embedded into HTML');
```

Testing The Full-Text Index

Trying the SQL command on our single row entry in the article table returns no results.

```
MySQL> SELECT * FROM article WHERE MATCH(author,
story) AGAINST('scripting');
```

You might be thinking, lets flame Leon... nothing is working! Before you do, you should keep in mind that MySQL will not return matched records if the word searched for is present in more than half the rows, thus a 50% threshold is used. Adding two more SQL inserts from Figure 2 will return the desired result.

To allow boolean full-text searching we must include IN BOOLEAN MODE into the AGAINST syntax:

```
MySQL> SELECT * FROM article WHERE MATCH(author,
story) AGAINST('+language' IN BOOLEAN MODE);
```

The SQL command returns all three entries from our article table. What went wrong? Why did all three rows return--and why wasn't the 50% threshold applied here? When searching in boolean mode, MySQL does not use the threshold feature, thus returning three out of three rows.

The MySQL documentation defines certain boolean operators one can use while searching. These operators are listed in Figure 3.

Figure 4 shows some SQLSQL examples of putting it all together.

Now that we have looked at MySQL as our search engine, let's focus on the methodology we would like to use in our PHP code.

Search Methodologies

When incorporating full-text search in our projects, it would be nice if we could separate the business logic

Figure 2

```
insert into article values(0, 'Tikvah Jesse Meyer', 'Notice how this is different from a script written
in another language like Perl or C -- instead of writing a program with lots of commands to output
HTML');
insert into article values(0, 'Hannah Lee Davids', 'What distinguishes the PHP language from something
like client-side JavaScript is that the code is executed on the server.');
```

Figure 3

Operator	Definition
+	A leading plus sign indicates that this word must be present in every row returned.
-	A leading minus sign indicates that this word must not be present in any row returned.
<>	These two operators are used to change a word's contribution to the relevance value that is assigned to a row.
()	Parentheses are used to group words into subexpressions.
~	A leading tilde acts as a negation operator, causing the word's contribution to the row relevance to be negative.
*	An asterisk is the truncation operator. Unlike the other operators, it should be appended to the word, not prepended.
"	The phrase, that is enclosed in double quotes ", matches only rows that contain this phrase literally, as it was typed.

Figure 4

```
mysql> SELECT * FROM article WHERE MATCH(author, story) AGAINST('+language -executed' IN BOOLEAN MODE);
mysql> SELECT * FROM article WHERE MATCH(author, story) AGAINST('"lots of commands"' IN BOOLEAN MODE);
```

from our specific implementation. This will create a true searching wrapper around full-text searching.

Here's a shopping list of business logic functionality we need:

- support for boolean operators in our queries
- ability to find the amount of articles that matched our entire query
- capability of paginating the results with a "next page" function
- support for caching the results of a query to allow the next results to free up MySQL processing time

To allow the separation of business logic and implementation, we can create a search class and a complimentary caching class to cache queries.

Getting To The Code

To start off we need to create an HTML form to allow us to search our articles database. The focus here is not on cosmetics as much as functionality (developers don't do design). Listing 1 shows a simple HTML form that calls our php script to perform the search. Listing 2 should be saved as search.php to allow for the form to work unchanged. We first include the searching class into our code . Notice the use of the $_GET system variable as register_globals in the php.ini file is turned off for security reasons. At this point you might wondering why I'm using a function to do the searching. The answer is simple: we use a function to facilitate the Next results feature.

The search function takes three variables as its arguments--first, the search string as entered from the query, second, the position in the results array to return, and third, the number of results to return. After creating a new instance of our search class, we use the count member variable in the class to check if any articles matched our query.

```
$results = new Search($search_for, $start,
                                    $increment);
if ( $results->count > 0 ) {
     // We found some articles
} else
     // We did not find any articles
}
```

The `$results->res` array contains the list of matched articles from `$start` to `$start + INCR`. Stepping through the array we are able to echo the articles found. Using

```
substr ($result['story'], 0, 128)
```

we display only the first 128 characters of our article. After all we do not want to display the whole article yet. I will leave it to you to add the functionality of clicking on the author to display the complete article.

The last part of the search function determines if we need to print a 'Next page...' link to view the rest of the search results. If `$results->count` is larger that the `$start + $increment` we have not come to the end of our results and we need to create a `$next` link to allow for continued searching. Note that we need to urlencode the `$search_for` text as it may contain special characters like a '+' that would be interpreted as a space in a URL instead of a plus sign.

As an enhancement to search.php, we could exclude the direct printing of results from the result loop and rather use a template system to separate the data layer from the presentation layer. We can also allow the author line to be click-able allowing the full article to be displayed, parsing the article_id as our reference.

The Search and Caching Class

Before we can start with our searching class we need to create a MySQL connection that we can use in the rest of our application. The following variables are used to connect to our articles database.

```
define("DATABASE",     'articles');
define("HOST", 'localhost');
define("USERNAME",     'www');
define("PASSWORD",     'www');
```

We need to create a new MySQL user that we can use for our web application with far less permissions. Normally a user with select, insert, update and delete functionality is all we need, allowing the user to only access the database from the localhost. One way of adding such a user is with the grant command.

```
mysql> grant select, insert, update, delete on
articles.* to www@localhost identified by 'www';
```

Remember that you have connect to the MySQL server with the administrative user for the grant command to work.

The following code will create a link to the server and connect to our articles database:

```
$link = MySQL_connect(HOST, USERNAME,PASSWORD);
if ( !$link )
  die("Could not connect to the MySQL server");
MySQL_select_db (DATABASE, $link);
```

Listing 1

```
<html>
<head>
<title>Search form</title>
</head>
<body bgcolor='#FFFFFF'>
<form method='get' action='search.php'>
<input type='hidden' name='function' value='search'>
<input type='hidden' name='start' value='1'>
Search for: <input type='text' name='search_for' size=20><br>
<input type='submit' value='Search'>
</form>
</body>
</html>
```

Listing 2

```
<html>
<head>
<title>Search Results</title>
</head>
<style>
td {
    font-family:        Verdana,Arial;
    font-size:          11px;
    color:              #100EB3;
     background-color:    #A0BFE5;
}
</style>
<body bgcolor='#FFFFFF'>
<table width=450 border=0 cellpadding=2 cellspacing=1>
<?php

// Include the searching class
// Make sure that the . path is included in your include_path
// variable of your php.ini file.
include("./Search.inc");

// The amount of results to return
define("INCR",     30);

// A search function to fascilitate Next results
function search($search_for, $start, $increment) {
    $results = new Search($search_for, $start, $increment);
    if ( $results->count > 0 ) {
        foreach ( $results->res as $result ) {
            echo "<tr><td valign=top>". $result['author'] ."</td>\n";
            echo "<td>". substr($result['story'], 0, 128) ." ...</td></tr>\n";
        }
        $new_start = $start + $increment;
        if ( $results->count >= $new_start ) {
            $encode_search = urlencode($search_for);
            $next = "<a href='search.php?function=search&search_for=$encode_search&sta
rt=$new_start&increment=". INCR ."'>Next results ....</a>\n";
        } else {
            $next = '';
        }
    } else {
        echo "<tr><td>No articles where found</td></tr>";
    }
    echo "<tr><td colspan=2>$next</td></tr>\n";
}

if ( $_GET['function'] == 'search' ) {
    search($_GET['search_for'], $_GET['start'], INCR);
}

?>
</table>
</body>
</html>
```

Let The Fun Begin

Our search class will take three parameters, the text we are searching for, the start of our return results and the number of results to include in our results array. Listing 3 shows the code of our searching class. Notice that we use an uppercase class definition, class Search, to allow us to easily distinguish at the time of instantiating the class, between normal functions and other classes.

First, we define some member variables we would use within the class to keep track of things. The most important variables are `$res` and `$count`. The `$res` variable will be used as an array containing the article information we need to return. The length of this array will be the value of the `$increment` variable (30 by default). The `$count` variable contains the total amount of articles that matched our query. After a match has been found the `$res` array will be populated by the id, author and story values of the article. The syntax of the array will be defined as:

```
$this->res[$this->count]['id'] = id that matched
$this->res[$this->count]['author'] = the author of
the article
$this->res[$this->count]['story'] = the full arti-
cle itself
```

Therefore we are creating an associative array using the counter `$count` as a place holder of amounts in our results array.

The constructor of the class sets the member variables and the following logic is followed:

1. Create a new instance of the cache class
2. If the search results have been cached
  2.1. Return the amount of entries found and a trimmed down SQL statement
  2.2. Use the SQL statement containing the id numbers of articles to build the results array $res
Else
  2.3. We create a full-text searching SQL statement using the MATCHED and AGAINST syntax
  2.4. We use the SQL statement to build the results array
  2.5. If we found some matched articles cache the ids of the matched articles
End

The return_sql function will return a valid SQL statement to use for searching our article table. If the `$search_for` variable is empty all the articles ordered by the author will be returned.

In the normal_search function we use the SQL statement returned from `return_sql` to query our database. Looping through the results array, we use the res_add function to add the id, author and story to our results array until we reach `$stop`. `$stop` is defined by `$start + $incr - 1`. As an example, start at entry 31 and return another 30, therefore stop adding items to the results array when we hit 60 but keep on counting the total amount of articles matched using `$count`. While counting the amount of articles matched, we build a `$matched` array containing the unique id numbers of the articles. We will therefore have the unique id number of every article that matched our search query.

How Does The Caching Work?

Listing 3 also contains the code for the caching class. We cache the query results by saving the unique id numbers of an article. A query will be cached if the search strings are different and if an hour has lapsed since the last query. A unique cache file is created by creating a md5 hash function using the `$search_for` text and today's date plus the current hour. In our example the date is created using the date function using today's date with the current hour as defined by the call to `date`('Ymd H'). For example:

```
$cache_file = md5("language"."20021119 11");
```

Using this method the query will be cached for a maximum of 59 min and a minimum of 1 min.

However if you would like for the query to be cached exactly 20 min, you can use the following code:

```
$new_time = strtotime("+20 minutes");
$new_date = date("Ymd Hi", $new_time);
$cache_file = md5($search_for.$new_date);
```

Our cached files are stored in the /tmp directory and have the .cache extension. If the filename created by the `md5` hash function with the extension .cache, exists in /tmp it means that we have cached the query and we should use the unique article ids in the cache file to build the `$res` results array. In the return_cache function we read the contents of the cache file into a variable and create an array from that variable using:

```
$cached = implode("", (@file($this->cache_file)));
$arr = explode(",", $cached);
```

To return the unique article ids, we use the `array_splice` function to return sub sections of the results array.

```
$result = array_splice($arr, $start, $inc);
```

Using the implode function on our results array we build the trimmed down SQL needed to return the articles in question.

Listing 3

```php
<?php

define("DATABASE",  'articles');
define("HOST",      'localhost');
define("USERNAME",  'root');
define("PASSWORD",  'root');
$link = mysql_connect(HOST, USERNAME, PASSWORD);
if ( !$link )
    die("Could not connect to the MySQL server");
mysql_select_db(DATABASE, $link);

class Search {
    var $stop           = "";
    var $start          = "";
    var $incr        = "";
    var $text           = "";
    var $res        = array();
    var $count          = 0;

    // Class constructor
    function Search($search_for, $start, $incr) {
        $this->start = $start;
        $this->incr = $incr;
        $this->stop = $start + $incr - 1;
        $this->text = $search_for;
        $cache = new Search_cache($this->text);
        if ( $cache->cached() ) {
            list($amount, $sql) = $cache->return_cache($start, $incr);
            $this->cache_search($amount, $sql);
        } else {
            $sql = $this->return_sql($this->text);
            $matched_ids = $this->normal_search($sql);
            if ( !empty($matched_ids) ) {
                $cache->store_cache($matched_ids);
            }
        }
    }

    // Return the correct SQL statement
    function return_sql($text) {
        $sql = '';
        if ( !empty($text) )
            $sql = " MATCH (author, story) against('$text' IN BOOLEAN MODE) ";

        if ( !empty($sql) )
            $query = " where $sql";
        return "select * from article $query order by author";
    }

    // Perform a normal search
    function normal_search($sql) {
        global $link;
        $res = mysql_query($sql, $link);
        while ( $row = mysql_fetch_array($res) ) {
            $this->count++;
            // Only return the amount we need
            if ( $this->count >= $this->start && $this->count <= $this->stop )
                $this->res_add($row);
            $matches[] = $row['id'];
        }
        if ( $this->count < $this->stop )
            $this->stop = $this->count;

        if ( is_array($matches) )
            $match_list = implode(",", $matches);
        else
            $match_list = '';
        return $match_list;
    }

    // Return the cached search results
```

Continued On Page 36...

Listing 3: Continued From Page 35...

```
72      function cache_search($amount, $sql) {
73          global $link;
74          $res = mysql_query($sql, $link);
75          while ( $row = mysql_fetch_array($res) ) {
76              $this->res_add($row);
77              $this->count++;
78          }
79          $this->count = $amount;
80      }
81
82      function res_add(&$row) {
83          $this->res[$this->count]["id"] = $row["id"];
84          $this->res[$this->count]["author"] = $row["author"];
85          $this->res[$this->count]["story"] = $row["story"];
86      }
87  }
88
89  // Manages the caching of queries
90  class Search_cache {
91      var $cache_file     = "";
92      var $timeout        = "Ymd H";
93
94      function Search_cache($text) {
95          $this->cache_file = '/tmp/'.md5($text.date($this->timeout)).'.cache';
96      }
97
98      // Check if a query has been cached
99      function cached() {
100         if ( is_file($this->cache_file) )
101             return true;
102         else
103             return false;
104     }
105
106     // If the query has been cached return the results
107     function return_cache($start, $inc) {
108         $start--;
109         $cached = implode("", (@file($this->cache_file)));
110         $arr = explode(",", $cached);
111         // Amount of matches found
112         $amount = count($arr);
113         $result = array_splice($arr, $start, $inc);
114         $sql = "select * from article where id IN (". implode(',', $result) .") order
by author";
115         return array($amount, $sql);
116     }
117
118     // Store the cached search ids
119     function store_cache($store) {
120         $fd = fopen($this->cache_file, "w+");
121         fputs($fd, $store, strlen($store));
122         fclose($fd);
123     }
124 }
125
126 ?>
```

Ultimately, we can even eliminate using the trimmed down SQL command to return cached entries and physically store the results in the cache file as an associative array with the unique id, author and story values. Using the PHP eval function we will be able to evaluate the information stored in the cache file.

Conclusion

We have looked in some detail at how to include a searching class using MySQL 4.0 full-text indexing capabilities into your projects. I hope that you have fun extending your applications and taking these examples to the next level.

To test the next results section, change the define("INCR", 30) line in search.php to define("INCR", 1) and search for the string 'language'. You should get three records back.

php|a

Leon is a developer based in Cresta, South Africa, who specializes in web application development.

EXCLUSIVE

The Zend of Computer Programming: Small Business Heaven

By Marco Tabini

This month, PHP powerhouse Zend Technologies launches a new program aimed at making its products more accessible to small businesses. We discuss its features, as well as the role of Zend in the PHP community, with President and CEO Doron Gerstel.

I think it's fair to say that, in the world of PHP, few people can say that they have never heard of Zend Technologies. After all, their co-founders wrote the Zend Engine, on which PHP itself has now been based for several years. Zend publishes several well known tools for PHP development, like the Zend Studio IDE or the Zend Performance Suite (which we reviewed in the December 2002 issue of php|a), and Zend.com is one of the best-known destinations for PHP-related information.

When we heard that Zend was working on a special offer for small businesses, therefore, our journalistic spider sense started twitching, and we sprang into action. What we discovered is an ambitious plan that will put the power of Zend's advanced tools in the hands of small businesses and allow them to take full advantage of PHP as they grow.

Products From the PHP Masters

The products that Zend offers are well known throughout the PHP community for reliability and effectiveness. The Zend Studio is a multi-platform IDE for Windows, Linux and MacOS that provides features like syntax highlighting, project management, a built-in debugger and integration with a server component for code optimization. Version 2.6, which is due out soon, will also include integration with the popular CVS (Concurrent Versioning System) version-control software. The Zend Encoder is a software tool that makes it possible to encrypt PHP scripts so that they are no longer readable by human but can still be executed by the PHP interpreter. Finally, the Zend Performance Suite provides a set of three different acceleration systems that increase the performance of PHP-based websites.

However, Zend's applications are not known for being inexpensive—the Enterprise Edition of the Zend Performance Suite alone starts from $1,875 (US), while the Encoder and Studio products can cost as much as $2,880 and $249 respectively. While these prices are probably easy to justify for a medium- to large-size organization, small businesses—the backbone of PHP adoption and usage—have traditionally been unlikely to adopt Zend's products en masse.

Small Business Heaven

All this is about to change, however. This month, Zend will announce a new program aimed at small businesses—defined as those companies whose annual revenues are below $250,000 (US)—that will make it significantly easier for them to acquire a great set of tools that are bound to increase their productivity in PHP-related activities, and we at php|a have the scoop.

The idea behind this initiative is very simple—businesses that qualify for the program will be eligible to receive a license for the Zend Studio, Encoder and Performance Suite for a one-time payment of $295 (US). The license for the Studio IDE is perpetual, while the Encoder and the Performance Suite are licensed for as long as a company remains within the program's criteria.

"PHP is in our bloodstream here at Zend, and we do whatever is necessary for the sake of this baby."

The potential implications of this initiative, however, are far-reaching. By allowing small businesses access to its products at such a low price, Zend is lowering the cost of providing high-quality PHP-based services for the entire community, thus ultimately improving the PHP market as a whole. Zend can clearly use this as a long-term strategy to promote the use of its products, not unlike what many other software companies have done for years. After all, businesses who participate in the program now are likely to remain faithful Zend users later on when they outgrow it.

However, this move makes a complete PHP-based solution—one that includes the protection of intellectual property, software optimization and caching, as well as a full-fledged development environment—one of the least expensive website development platforms available, which, in turn, provides small businesses with an opportunity to be highly competitive in the current economy.

Q&A With High Up Above

With such news afoot in the wind, we could not resist but exchange a few words with the head honcho at Zend. No, we're not talking about Zeev Suraski, although he is probably the most widely recognized member of the Zend team and its co-founder.

Doron Gerstel - President and CEO of Zend

The focus of our interview is Doron Gerstel, who cofounded Zend and is its President and CEO. At 42 years of age, Mr. Gerstel has been around the block a few times, holding positions at ESC Medical Systems and Fibronics Ltd., and sitting on the board of directors of companies like Fritz Inc., America's third-large logistics provider, which was acquired by UPS in February of 2001.

We asked Doron questions on Zend, PHP and the small business program that Zend has just launched:

php|architect: What is Zend's mission?

Doron Gerstel: The answer to this question is very simple. From Zend's first days, our mission has been to give solid backing to the PHP market, through direct open-source activities as well as through our full spectrum of development, protection and scaling products.

php|a: Which role would you like Zend to assume within the PHP community?

DG: Our role in the PHP world consists of a number of responsibilities. First of all, Zend continues to work at development of the Zend Engine and the PHP project, as well as general PHP evangelism and education. The PHP community has come to expect this, we enjoy this work, and it is necessary for PHP's continued growth.

Zend is lowering the cost of providing high-quality PHP-based services for the entire community.

Second, Zend's R&D efforts will continue to bring innovative products. We are often considered the 'bell-weather' or measuring stick by which professional products are measured, and I intend for this continue with our current products as well as new products that we'll be coming out with in 2003.

php|a: How many of its resources does Zend dedicate to the PHP development effort?

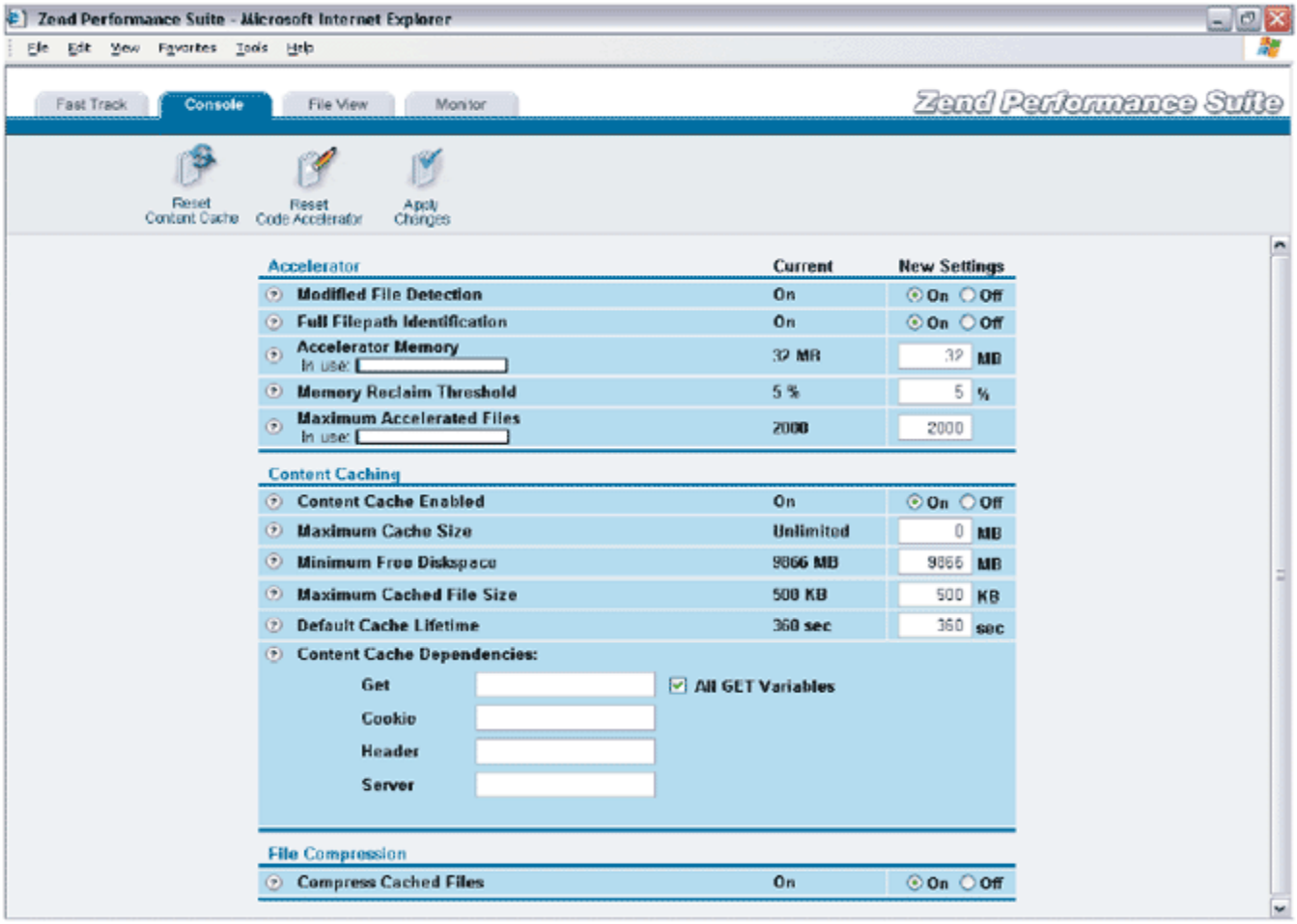
DG: We have three core engine developers that are very actively involved, as well as supplemental development and QA teams that pitch in during peak development times.

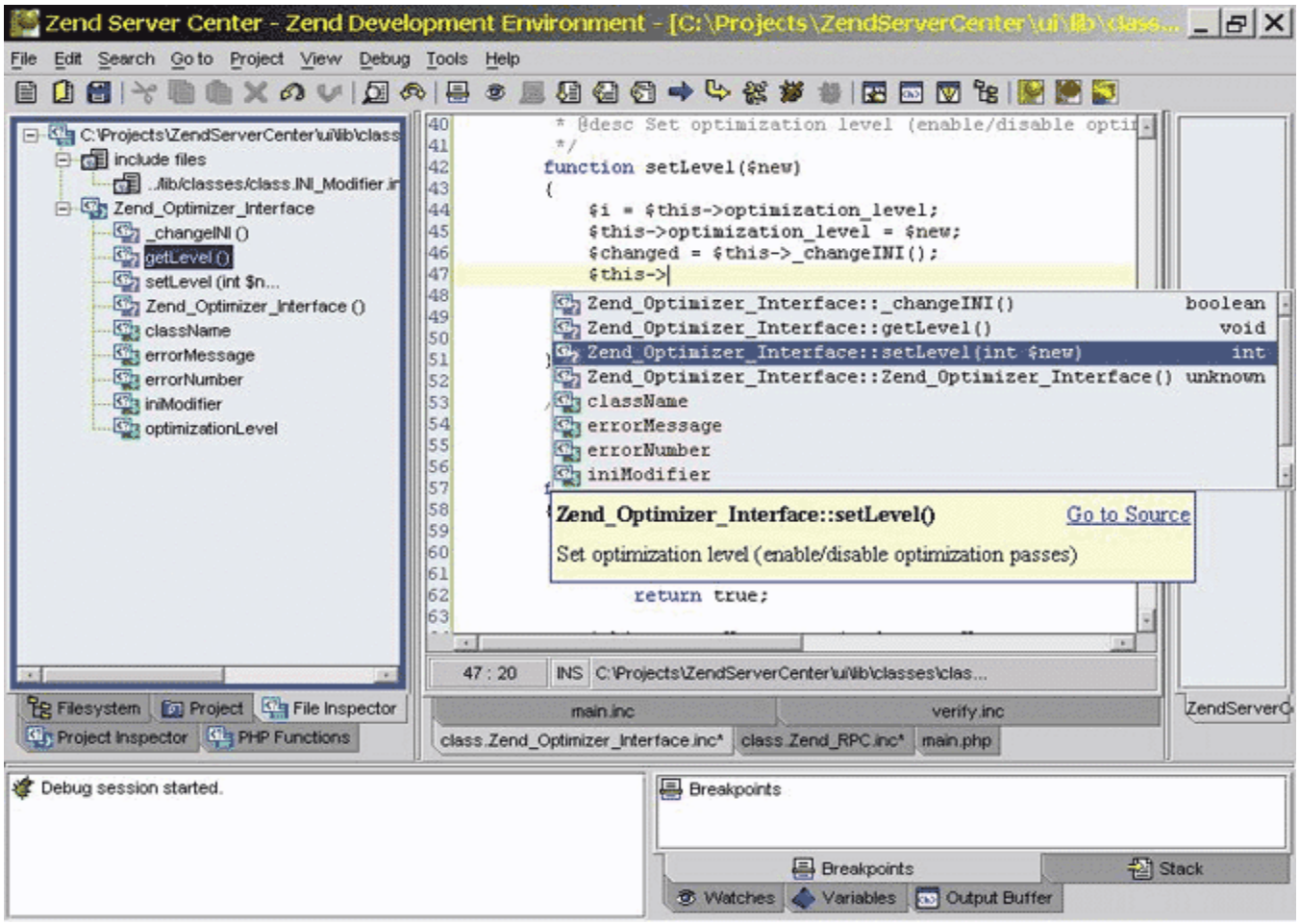
It's worth noting also that our commitment to the PHP community doesn't end with just Zend Engine / PHP development. Our Zend.com Developer Zone also takes significant editorial resources, which we do for the sake of promoting PHP usage.

php|a: How do you determine this number?

DG: PHP is in our blood-stream here at Zend, and we do whatever is necessary for the sake of this baby. So, the core PHP development team activity is untouchable.

Beyond this, we grow our investment almost exactly in proportion with our revenue growth. As more customers choose Zend, Zend invests more into the com-





munity. This is something I'm very proud of, especially when I look at some of the other commercial vendors who sell PHP products.

php|a: Do you think that PHP needs a more structured development approach and more business focus, like Java? Have you ever considered taking a more active approach to this end?

DG: I'll split that question into two - Structured Development and Business Focus.

Regarding structured development, only a bit. If you look at the adoption rate, the product head-to-head comparison, and the rate of product feature advancement, you'll see that PHP really shines. The whole PHP Development Team should be very proud of this fact. That said, a bit more structure might help. For example, PHP 5 could have been available by now if we had more structure in the development process.

About the business focus, here there is no question. PHP does not have a strong evangelism body, and the result is felt by all of us. Despite the adoption, we all still fight for even a fraction of the awareness that JSP and ASP get. Can you even imagine what the adoption rate of PHP would be today if PHP had the marketing force behind it that Java has?

Zend will definitely be growing its efforts for PHP evangelism, but this won't be a solo effort.

php|a: Which organizations do you think benefit the most from the use of PHP?

DG: The type of organization that benefits from PHP is, frankly, the type that focuses on results as opposed to hype. In terms of size, large organizations benefit from PHP's scalability, both in terms of price as well as power. Smaller organizations benefit from ease of deployment and low start up cost. This, by the way, was the impetus to introduce our Small Business Program.

php|a: Tell us about the new small-business initiative. What do you hope to accomplish by introducing it?

DG: Our Small Business Program provides a full collection of Zend products, at a price that is very amenable to start-ups or small companies that typically have to worry about

cash-flow issues.

Zend itself grew from being a living-room based start-up. I see the number of PHP-powered small companies, so many of which have unique talents and innovative business concepts, and I want to give the assistance that I am capable of providing.

From a commercial benefit point of view, it's a long-term strategy. By adding one more pillar of support to the emerging PHP market, we hope that Zend's market will thus grow in the future. The short-term benefit is negligible.

php|a: Are you afraid that you will dilute the perceived value of your products by offering them at a reduced price?

DG: No, I don't think so. Our products provide tremendous value. Our customers tell us this over and over.

The fact that Zend is helping many more small businesses get through their early growth stages does not diminish the value that the products offer. Look at Zend Encoder, for example. Even the smallest PHP organizations recognize that intellectual property protection is mission-critical, and that it has tremendous value in terms of the revenue that it can generate for them in the future. Not to mention the productivity gained via Zend Studio, which is integrated with Zend Encoder. The problem is that, still being small, they need to test the market and prove that the revenue will come.

Interestingly enough, Zend has always offered a significant educational discount, for universities, schools, or any educational/research organization. This did not take away from perceived value. In this sense, I believe that the Small Business Program too will reinforce the value, not dilute it.

php|a: Will the level of customer support you will offer to businesses who will participate in this initiative be the same as what you offer to your regular customers?

DG: Absolutely.

php|a: Do you plan to provide any type of marketing support to Zend's small-business customers?

DG: Yes, we do. Zend is most known for its products, and thus the products are the star attractions of the Small Business Program. But smaller companies also need help in getting more marketing exposure and business development, and with the tremendous traffic at Zend.com, we have the ability to provide this, to everyone's benefit.

php|a

The Zend Small Business Program
All the Details

The SBP is for all business whose annual revenues are below $250,000 (US). Companies that meet the requirements receive the following:

Zend Studio 2.0 — Zend's PHP editor includes a host of features designed to make the creation of PHP-based applications easier.

Zend Performance Suite — This server-side tool integrates directly with Apache 1.3 to provide additional scalability for your application. It includes:

- A code compilation tool, designed to reduce the time that the PHP engine takes to parse your script

- A GZIP compression system that makes your pages load faster by sending them in compressed format to the user's browser

- A content-caching mechanism that accelerates your site's performance by creating static versions of its pages

Zend Encoder — A system that allows you encrypt your scripts so that they cannot be viewed by any parties you redistribute them to. It can be useful for protecting your intellectual property and ensuring that those techniques and algorithms your team worked so hard on cannot be just copied and reused by someone else.

All the programs that are part of the SBP are licensed for as long as the customer meets the requirements of the program, with the exception of the Zend Studio product, which comes with a perpetual license. A support plan that includes upgrades to the key components is also available.

FEATURES

Using The .NET Assembly through COM in PHP

By Jayesh Jain

Microsoft is pushing .NET as the next wave of web development for the Windows platform (and, indeed, for the Internet in general through its Shared Code "Rotor" initiative). While PHP cannot yet work natively within the .NET framework, Windows users have the alternative to interact with it through COM.

The .NET Framework is a new computing platform that aims to simplify application development in the highly distributed environment of the Internet. It is designed to provide a consistent object-oriented programming environment, as well as a code execution environment. With these features, .NET aims to minimize software deployment and versioning conflicts, guarantee safe execution of code, and build all communication on industry standards to ensure that code based on the .NET Framework can integrate with any other code.

Here are few advatages of the.NET framework:

Improved Reliability
Increased Performance
Developer Productivity
Powerful, Granular Security
Integration with Existing Systems
Ease of Deployment
Mobility Support
Native XML Web Service Support
Support for over 20 Programming Languages
Flexible Data Access

.NET Assemblies and COM

An assembly is the primary building block of a .NET Framework application. It is a collection of functionality that is built, versioned and deployed as a single implementation unit containing one or more files. Each assembly contains an assembly manifest, a collection of metadata that describes the relationships between each of the elements that are part of the assembly itself.

COM (Component Object Model), arguable the most popular component software models available on Microsoft platforms, defines how objects expose themselves for use with other objects and how processes communicate across the network. Because COM objects are reusable binary components, they are easily integrated with Visual Basic, PHP, Java or any other language that supports this standard. COM provides users with a wide range of services, easy to use tools and hundreds of applications and components.

What is COM Interop?

You have to understand from the outset that .NET and COM are basically two different technologies,

REQUIREMENTS

PHP Version: **4.0 and above**
O/S: **Windows**
Additional Software: **.NET Framework**

which, unfortunately, are not compatible with each other.

However, Microsoft realized that, given the investment people have made over the years in COM technology, providing some way of utilizing existing COM applications within the .NET Framework was necessary—this is how the concept of "COM Interoperability", or COM interop, came to be.

COM interop allows you to create a COM wrapper around .NET components, which makes Windows look at them as COM objects, thus making them available to any calling COM applications like any standard COM based component.

COM functionality is available only to the Windows version of PHP

COM and PHP

PHP has built-in functions for using COM objects. Using the COM Interop feature of .NET, we'll create a wrapper around a .NET assembly and use it in PHP (in fact you could use the same steps to use a .NET assembly with VB6 or any other COM-compatible application).

Before you go any further, I want to note that COM functionality is available only to the Windows version of PHP. This may sound quite obvious, since COM is a Windows-only technology, but Microsoft has announced that it intends to make .NET a multi-platform technology and this could lead to some confusion.

As long as you're running PHP in a Windows environment, therefore, you do not have to install any additional software over PHP to use COM, though you'll want to be aware of a few settings in the php.ini (shown in Figure 1) which can affect COM functionality in PHP.

This article assumes that you have a fair knowledge of the .NET Framework, PHP and COM, and that you have Visual Studio .NET, PHP and IIS or Apache installed and working properly on your computer.

Creating a .NET Assembly

Lets start our tutorial by creating a very simple .NET assembly. To create the assembly we will use Visual Studio and the Visual Basic language. Keep in mind, though, that you could in fact use C# or any other languages supported by the .NET architecture.

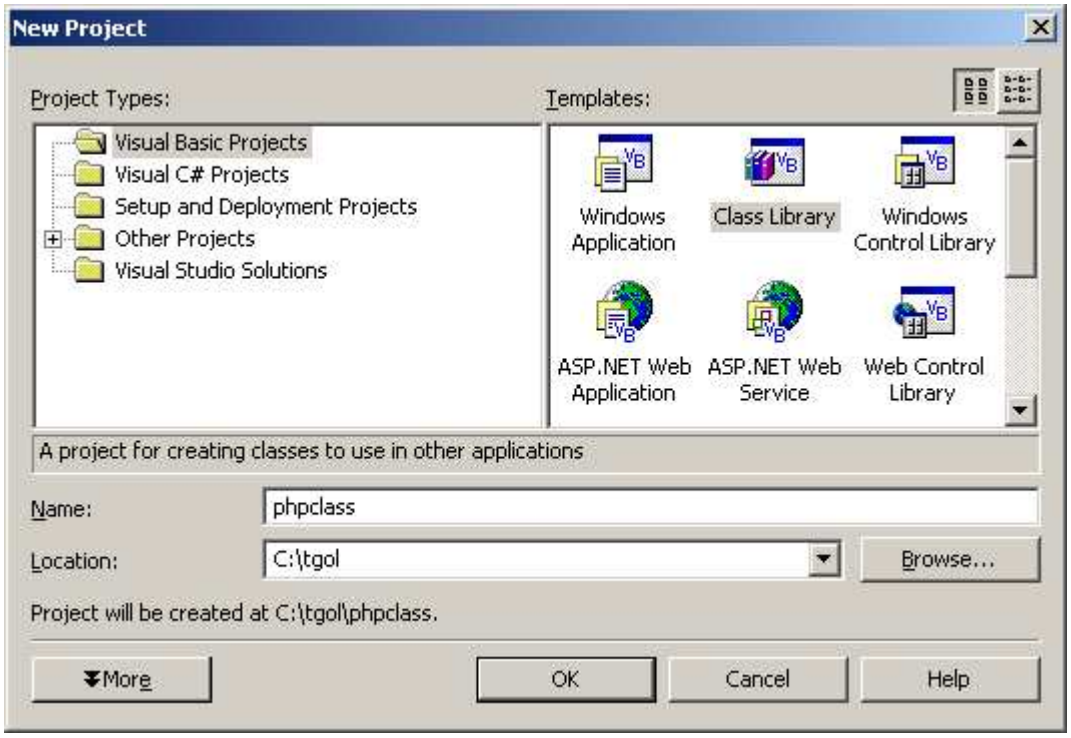
To keep the example short and sweet, let's create a very simple class called patient, with two properties: LMP (Last Menstrual Period) and EDD (Estimated Delivery Date). We want the ability to set and read lmp, and the other property, edd, will be calculated from the lmp property.

We will use the formula 'edd = lmp + 280 days' (this is a standard calculation for human gestation period), to derive a value for our edd property.

Figure 1 - php.ini settings

Setting Name and Description	Default Value
com.allow_dcom Set to one if DCOM functionality is to be allowed. This setting can be used to limit the use of components that are not on the local machine	0
com.autoregister_typelib Set to 1 if you want PHP to automatically register the type libraries specified in the com.typelib_file INI entry	0
com.autoregister_verbose Set to 1 if you want PHP to print out additional information while registering typelibs	0
com.autoregister_casesensitive Set to 1 if you want type libraries to be registered in a case sensitive way	1
com.typelib_file Defines a file that contains an arbitrary number of type libraries that PHP can preload into its namespace so that it will recognize constants and type definitions	""

Figure 2 - Creating a new class in Visual Studio .NET



Let's see all of these things in action now. Open Visual Studio.Net and create a new class library project, called "phpclass". Click OK to create this project, and Visual Studio creates all the file(s) necessary to create this project (Figure 2).

Visual Studio creates a class (class1) for you to start with. Open the code window for class1 (class1.vb), if it is not open already, by double clicking the class1.vb file in the Solution Explorer. Type the code in Listing 1 into the code window for class1 (make sure you delete all the default code genereated by Visual Studio, since we are replacing it in its entirety).

This VB.NET code calculates the EDD (Estimated Delivery Date) from the specified LMP (Last Menstrual Period) date for any patient. To implement this, we have created a namespace called HealthRecord and a public class patient with two public properties, lmp and edd. The edd property is a readonly property, which is calculated from lmp which is a read-write property.

We have used the format function to format the date variable to human readable format. We've also used the dateadd function to calculate the edd, which is 280

Listing 1 - Code for Class1.vb

```
Namespace HealthRecord
    Public Class patient
        Private m_lmp As Date
        Public Property lmp()
            Get
                Return Format(m_lmp, "D")
            End Get
            Set(ByVal Value)
                m_lmp = Value
            End Set
        End Property
        Public ReadOnly Property edd()
            Get
              'EDD is 280 days from LMP
              Return Format(DateAdd(DateInterval.Day, 280, m_lmp), "D")
            End Get
        End Property
    End Class
End Namespace
```

Figure 3 - Some options for the sn.exe tool

Option	Description
-D *assembly1 assembly2*	Verifies that two assemblies differ only by signature. This is often used as a check after an assembly has been re-signed with a different key pair.
-e *assembly outfile*	Extracts the public key from assembly and stores it in outfile.
-h	Displays command syntax and options for the tool.
-i *infile container*	Installs the key pair from infile in the specified key container. The key container resides in the strong name CSP.
-k *outfile*	Generates a new key pair and writes it to the specified file.
-m [y|n]	Specifies whether key containers are computer specific, or user specific. If you specify y, key containers are computer specific. If you specify n, key containers are user specific. If neither y nor n is specified, this option displays the current setting.
-o *infile [outfile]*	Extracts the public key from the infile and stores it in a .csv file. A comma separates each byte of the public key. This format is useful for hard coding references to keys as initialized arrays in source code. If you do not specify an outfile, this option places the output on the Clipboard.
-q[uiet]	Specifies quiet mode; suppresses the display of success messages.
-?	Displays command syntax and options for the tool.

days from the lmp.

Creating a Strong Name Key file

A 'Strong Name' is the full class name with namespace, version, public key and digital signature. All of the assemblies installed on the system must be uniquely identified and versioned. To make sure the assembly is not tampered with, it also needs to be signed with a common key (public key) for decryption.

The Strong Name command-line tool (sn.exe) can be used to generate a new public-private key pair and to write that pair to a file which could then be used to create the assembly.

Figure 4 - Output of the strong name generation utility

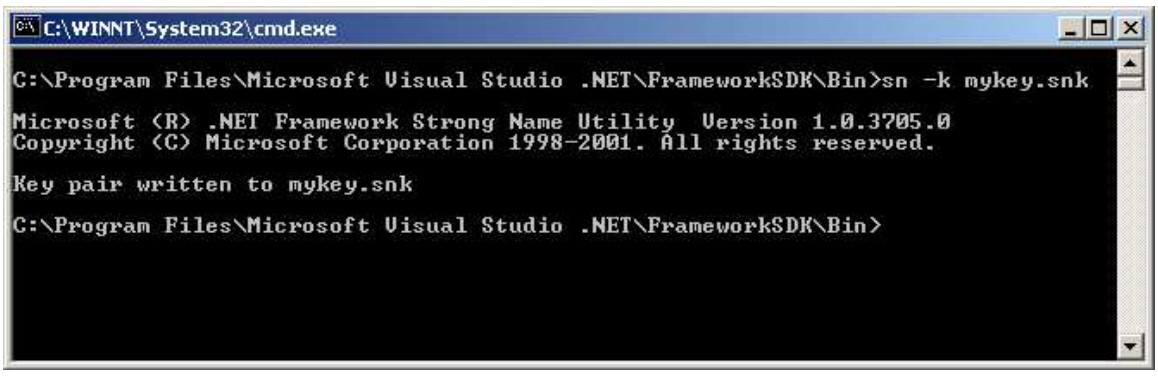


Figure 5 - Some options for Assembly Registration Tool

Option	Description
/help	Displays command syntax and options for the tool.
/nologo	Suppresses the Microsoft startup banner display.
/regfile *[:regFile]*	Generates the specified .reg file for the assembly, which contains the needed registry entries. Specifying this option does not change the registry. You cannot use this option with the **/u** or **/tlb** options.
/silent or **/s**	Suppresses the display of success messages.
/tlb *[:typeLibFile]*	Generates a type library from the specified assembly containing definitions of the accessible types defined within the assembly.
/unregister or **/u**	Unregisters the creatable classes found in *assemblyFile*. Omitting this option causes Regasm.exe to register the creatable classes in the assembly.

Figure 3 shows some of the options available to the sn.exe tool.

```
sn [-quiet][option [parameter(s)]]
```

To generate the Strong Name Key File we run the following command:

```
sn -k mykey.snk
```

This utility can be found in the .NET Framework's Bin folder `C:\Program Files\Microsoft Visual Studio .NET\FrameworkSDK\Bin` (this might be different on your computer). Alternatively, you could run a .NET Command Window by selecting Start –> Programs -> Microsoft Visual Studio .NET -> Visual Studio .NET Tools >- Visual Studio .NET Command Prompt, which sets all of the required paths for you.

Figure 4 shows the output of the strong name generation utility that you should be seeing.

```
<Assembly:
AssemblyKeyFile("c:\tgol\phpclass\mykey.snk")>
```

Adding the Strong Key to Project

We need to add the key we just created to our project, so open Solution Explorer and double click AssemblyInfo.vb to open the code window and add the following:

Please make sure you enter the full path where you have saved the key file.

Now Build the Application to create phpclass.dll (this file will be created in your project's bin folder) `c:\tgol\phpclass\bin\phpclass.dll` (this can be different on your computer).

Figure 6 - Assembly registration output

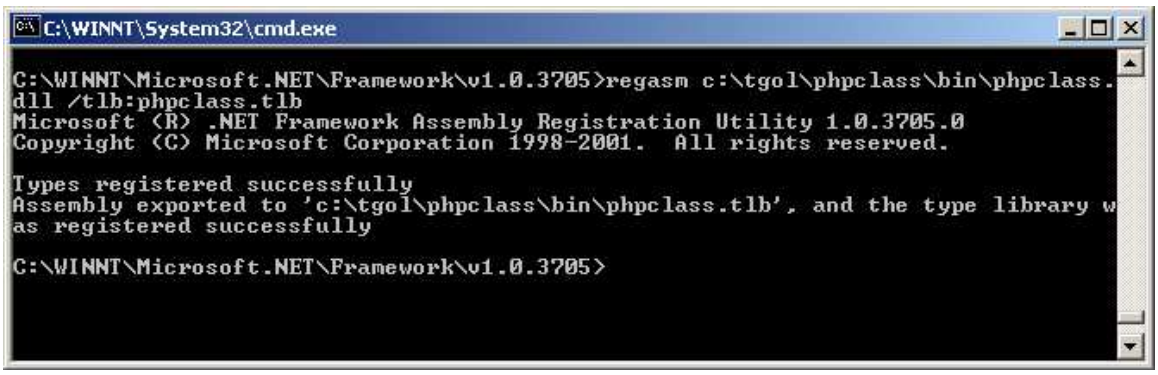


Figure 7 - Some options for the Global Assembly Cache tool (gacutil.exe)

Option	Description
/cdl	Deletes the contents of the download cache.
/h[elp]	Displays command syntax and options for the tool.
/i *assemply*	Installs an assembly into the global assembly cache.
/l	Lists the contents of the global assembly cache.
/ldl	Lists the contents of the downloaded files cache.
/nologo	Suppresses the Microsoft startup banner display.
/silent	Suppresses the display of all output.
/u[ngen] *assembly*	Uninstalls a specified assembly from the global assembly cache. If you specify /ungen, Gacutil.exe also removes the assembly from the native image cache. This cache stores the native images for assemblies that have been created using the Native Image Generator (Ngen.exe).
/?	Displays command syntax and options for the tool.

Registering the Assembly

To register a .NET class with COM, you must run a command-line tool called the Assembly Registration Tool (regasm.exe). The Assembly Registration tool reads the metadata within an assembly and adds the necessary entries to the registry, which allows COM clients to create .NET Framework classes transparently. Once a class is registered, any COM client can use it as though the class were a COM class. The class is registered only once, when the assembly is installed. Instances of classes within the assembly cannot be created from COM until they are actually registered.

The syntax for the Assembly Registration Tool is as follows:

```
regasm assemblyfile [options]
```

Some of its options are shown in Figure 5.

The regasm utility can be found in `C:\WINNT\Microsoft.NET\Framework\v1.0.3705` (although this might be different on your computer, depending on where you have installed the .NET Framework).

To register our assembly we'll run the following command, whose output is shown in Figure 5.

```
regasm c:\tgol\phpclass\bin\phpclass.dll
/tlb:phpclass.tlb
```

Adding an Assembly to the Global Assembly Cache

Each computer where the common language runtime is installed has a machine-wide code cache called the global assembly cache. The global assembly cache stores assemblies specifically designated to be shared by several applications on that particular computer.

The Global Assembly Cache tool (gacutil.exe) allows you to view and manipulate the contents of the global assembly cache, as well as the download cache. You can invoke gacutil using the following syntax:

```
gacutil [options] [assembly]
```

Listing 2 - Instantiates a copy of Microsoft Word and shows it on your local desktop

```
1  <?php
2
3 // Create COM object for Microsoft Word
4 $myword = new COM("word.application") or die("Cannnot Open Microsoft Word");
5 // Display Microsoft Word Version
6 echo "We have loaded Version $myword->Version of Microsoft Word<BR>";
7 // Open Microsoft Word
8 $myword->Visible = 1;
9
10 ?>
```

Listing 3 - phpnet.php

```
<?php

$MyObj = new COM ("phpclass.HealthRecord.patient");
$MyObj->lmp = "05/08/2002";
echo "LMP : $MyObj->lmp";
echo "<BR>EDD : $MyObj->edd";

?>
```

Its options are shown in Figure 7.

To add our assembly to the GAC we shall run the following command:

```
gacutil /i c:\tgol\phpclass\bin\phpclass.dll
```

This utility can be found in `C:\Program Files\Microsoft Visual Studio .NET\FrameworkSDK\Bin` (again, this might be different on your computer). The results of this operation call should be similar to what shown in Figure 7.

To view all of the assemblies installed in the GAC on your computer, open your Windows Explorer and look in the assembly folder under your windows folder `C:\WINNT\assembly` (on my computer).

There is another way to add an assembly to the GAC: by using Microsoft Windows Installer 2.0. This is the recommended and most common way as Windows Installer provides reference counting of assemblies in the global assembly cache, as well as other benefits.

How to use COM in PHP

A COM class provides a framework to integrate COM or DCOM components into your php scripts.

To use the COM functionality we have to instantiate an object of type "COM" by using the following syntax:

```
COM (string module_name [, string server_name])
```

Let's go over the parameters we can use:

a) **module_name** This is the name/classid of the component to be used (Please do keep in mind that to use any COM object, it has to be regitered and should have a valid entry in the Windows registry).

b) **server_name** This is the name of the Distributed COM server from which the component should be used. This is an option parameter and if not specified localhost is assumed. (if you want to use DCOM, you must ensure that the DCOM com.allow_dcom setting has has been set to true in php.ini.)

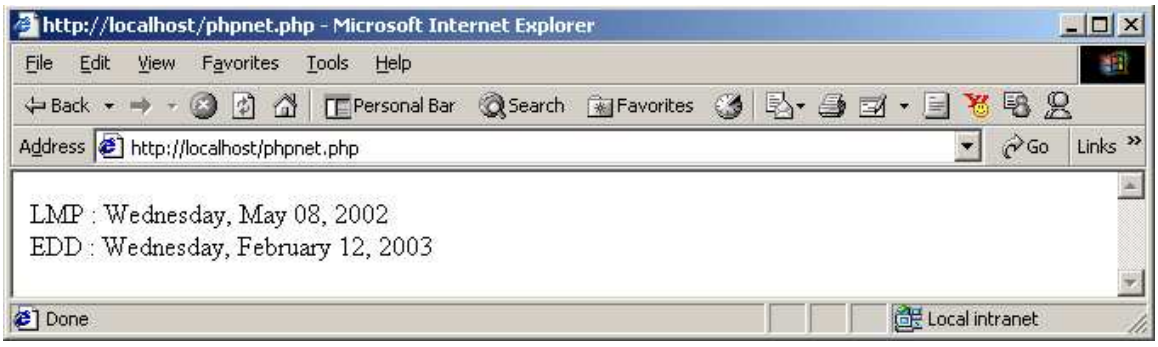
Once instantiated, the object assumes all the properties and methods of the corresponding COM class, so that it can be used as if we had just instantiated the COM class itself. Pretty neat, eh?

Now that I have introduced the COM framework provided by PHP, let's see some COM implementation in action. Thanks to the fact that there are hundreds of COM applications available, we can test the COM functionality of PHP with ease.

In fact, I'd be willing to bet that at least one COM application is installed on your machine (assuming, of course, that you run Windows!). Microsoft Word, Microsoft Powerpoint, Microsoft Outlook, Microsoft Excel are some example of COM-compatible applications.

A simple PHP-COM script could be similar to what is shown in Listing 2. You can execute it directly from the command line (using the CLI version of PHP) or from within your web browser, although it doesn't make much sense to do so in this case, since the listing instantiates a copy of Microsoft Word and shows it on your local desktop.

Figure 8 - Output for phpnet.php



Coding the PHP File to use our .NET Assembly

Now that we know how to use COM functionality in PHP, we will write a PHP script to use the COM wrapper we just created over the .NET assembly.

Create a text file with the following content and save the file as phpnet.php in your webserver's root folder.

As you can see, our first step consists of creating a COM object for the class we just installed and registered. Since COM interop makes the .NET assembly look like a COM object, we can use the same syntax we used to create COM objects in PHP. Please note that the class name depends on the project name, namespace and the class name used to create the .NET assembly.

Next, we set the lmp property to a valid date, which will be used to calculate edd. Finally, we can output the LMP property in human readable date format and also output the calculated EDD property to the browser.

If you open your favorite browser and type http://localhost/phpnet.php to see the PHP file in action, the output should look similar to Figure 8.

Cleaning Up

I doubt that you will want to keep the .NET class we've created on your hard drive, so let's take a look at what the procedure for removing it is. Because of all the references created by the installation process, you can't simply delete the object files from your hard drive.

The first step consists of unregistering the class' type library by calling regasm with the /unregister switch and the assembly path:

```
regasm /unregister c:/tgol/phpclass/bin/php-
class.dll
```

Next, you will need to remove the assembly from the GAC by running gacutil with /u switch and the assembly name.

```
gacutil /u phpclass
```

In both cases, it's important not to specify the path to the assembly and to omit the .dll extension, since regasm and gacutil will be able to find those out by themselves based on the information stored in the registry.

The .NET Framework SDK also provides a Windows shell extension called the Assembly Cache Viewer (which is part of Shfusion.dll), which you can use to remove assemblies from the global assembly cache.

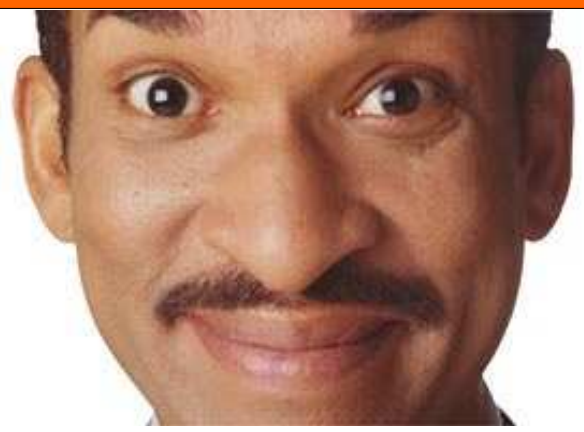
Conclusion

As you have seen, making PHP speak to .NET through COM is easy and convenient. Even though the example I provide here is trivial, it demonstrates that COM Interop is a viable tool for enhancing any application by taking advantage of literally thousands of COM objects available on the market. Naturally, using COM will limit your PHP code to properly functioning only under Windows—but if that is a limitation you're willing to live with, then it's worth looking into what it has to offer.

php|a

Jayesh Jain is an Internet consultant based in New Zealand. We asked him to expand on his previously published .NET COM Interop work, and he very kindly obliged by writing this article. You can reach him at jayesh74@yahoo.com

Connect with your database



Publish your data fast with PHPLens

PHPLens is the fastest rapid application tool you can find for publishing your databases and creating sophisticated web applications. Here's what a satisfied customer, Ajit Dixit of Shreya Life Sciences Private Ltd has to say:

*I have written more than 650 programs and have almost covered 70% of MIS, Collaboration, Project Management, Workflow based system just in two months. This was only possible due to **PHPLens**. You can develop high quality programs at the speed of thinking with **PHPLens***

Visit **phplens.com** for more details. **Free download**.

FEATURES

Writing Secure PHP Code

By Theo Spears

When people ask me to look at their sites, the most common problem I come across is a complete lack of security. Sometimes this is because they have not thought about making the site secure, and other times because they have tried to do so but weren't aware of all the possible ways a hacker can compromise a site. Either way I often get asked by people why they should care about security - after all, who would want to hack their site?

People who are on the security bandwagon would laugh at that and reply with something like "but of course you should make it secure, there are hundreds of teenage hackers around with nothing better to do than try to hack websites." In spite of this it is a good question. Why bother spending 20 hours making your site secure if you can recover from a compromise in 1 hour? Why bother focusing on the security of a tiny project that will only ever be used on your computer?

I can't tell you that you must make all your projects secure—that is for you to decide. I will, however, tell you why I focus on the security aspect of everything I do on the web, even for something as simple as a personal diary. By focusing on the security of small projects I find I get into the habit of programming securely. At first, I thought it was a real pain, but now I do it automatically. The more you try to write secure code, the more you start writing it without thinking.

Security guides often concentrate almost entirely on theory and focus very little on how things should actually be done

Security guides often concentrate almost entirely on theory and focus very little on how things should actually be done. Unfortunately, security isn't something you can learn by theory alone, so while I will discuss theory from time to time, for now let's write some code and then explain how it can be broken.

A Bug or a Feature?

One of the features of PHP that made it convenient for writing web scripts is that in older versions if you had a form with a textbox called 'foo' and the user typed 'bar' into it the variable $foo in your script would automagically be set to 'bar'. This feature is known as register globals, and it has caused countless headaches for programmers trying to write secure scripts.

REQUIREMENTS

PHP Version: **4.0 and above**
O/S: **Any**
Additional Software: **N/A**

A Typical Mistake

Remember the diary I mentioned earlier? I'm going to show you part of the code from it now as an example of some fairly typical code. Unfortunately, the code is not secure. When I wrote the diary about a year ago it was only for me so I didn't bother with a database or anything complicated like that. I just hardcoded the values right into the file. My login code looked something like this:

```
if ( $username == 'Grendels'
        && $password == 'xxxxxxxx' ) {
  $logged_in = true;
} else {
  show_login_form ();

}

...
if ( $logged_in == true ) {
  show_diary_entries ();
}
```

(Before you ask, xxxxxxxx wasn't really my password). So, what is wrong with that code? To the untrained eye, it looks safe enough—if the user enters a bad username or password, it will ask them to login again. However, it's far from safe, and to better understand its limitations, we need to look at what happens when someone tries to login.

To simplify logging into my diary I had a bookmark that automatically entered my username and password for me. It looked like this:

```
http://localhost/diary.PHP?username=Grendels&pass-
                    word=xxxxxxxx.
```

When I visited that link $username was set to 'Grendels' (from the 'username=' part) and `$password` was set to 'xxxxxxxx' (from the 'password=' part).

If you still don't see a problem, imagine if I now visited `http://localhost/diary.PHP?logged_in=true`. There is no username or password so the login part of the code works fine and shows a form asking you to login. But wait, what happens in the next bit of code? In spite of not being logged in, `$logged_in` is actually set to 'true'. This is because it has been set from the url in the same way the username and password were set in the earlier example. Suddenly all my inner secrets have been revealed to the world!

The Evils of Register Globals

When register globals is switched on, any values the user sends are automatically converted into a variable. The three primary sources of these variables are forms the user fills in, variables added onto the end of the url (like in the example above) and cookies that your scripts have set. It was very convenient to use globals in earlier PHP versions, where the alternative was to use `$HTTP_GET_VARS`, `$HTTP_POST_VARS` and `$HTTP_COOKIE_VARS`. I have to confess I never bothered to use these much longer variable names.

As you can see in my examples above, the (now deprecated) 'register_globals' feature of PHP can make securing even very simple web programs difficult, and the results can be far more severe than just allowing other people to read your diary. Unfortunately, in spite of the known security issues, many web servers still have it switched on and many scripts still rely on it.

To address this problem, since PHP 4.1 there are now much abbreviated variables to replace the long ones above. They are called superglobals and include `$_GET`, `$_POST` and `$_COOKIE`. They are shorter to type and you can access them easily from functions. If you are currently using variables like `$username` for parameters passed from the browser you should visit www.PHP.net and look up more about the superglobals. The techniques we will review below will keep code that relies on 'register_globals' from endangering your site - in other words, crackers will no longer be able to set `$logged_in=true`.

Method 1 : Turn Off Register Globals

By far the simplest and possibly the safest way to avoid problems caused by `register_globals` is simply to turn it off. If you look in your PHP.ini file you will see a line that starts `register_globals = `. Change this to `register_globals = off` and you can be fairly certain you are safe.

This may be the most simple and obvious method, but I don't like it. It is fine on your own server, but if you ever put your scripts on another server, you may not be given a choice. The majority of PHP Internet hosts, for example, have register globals turned on, so turning this option off is only an option if you own the server your website will be running from.

Method 2 : Unregister the Globals

As we cannot reliably stop global variables from being set, the most direct option is to unset them again. The code to unset all global variables is just a simple loop, as shown below.

```
foreach ( $GLOBALS as $key => $value ) {
  if ( $key != 'GLOBALS' && $key != 'key'
                && $key != 'value' ) {
            unset ( $GLOBALS [ $key ] );
        }
      }
unset ( $key, $value );
```

The $GLOBALS array contains one key/value pair for each global variable (including itself). Unsetting the key in $GLOBALS also unsets the equivalent global variable. The 'if' block is to keep the $GLOBALS array from unsetting itself, which could have a nasty effect on other parts of your script. It also prevents an infinite loop by excluding $key and $value until afterwards.

If you test this with your script you will find you can no longer receive any input from the user. This is because the script resets all global variables, including $_GET, $_POST, etc. This is clearly not what we want - a script that can't receive input from the user is pretty useless. The solution is to first save a copy of each of the superglobals inside a function and then write these copies back after the loop has run. I have provided a handy function that does all this, which is shown here:

```
function unregister_globals () {
  $REQUEST = $_REQUEST;
  $GET = $_GET;
  $POST = $_POST;
  $COOKIE = $_COOKIE;
  if ( isset ( $_SESSION ) ) {
    $SESSION = $_SESSION;
  }
  $FILES = $_FILES;
  $ENV = $_ENV;
  $SERVER = $_SERVER;

  foreach ( $GLOBALS as $key => $value ) {
    if ( $key != 'GLOBALS' ) {
      unset ( $GLOBALS [ $key ] );
    }
  }

  $_REQUEST = $REQUEST;
  $_GET = $GET;
  $_POST = $POST;
  $_COOKIE = $COOKIE;
  if ( isset ( $SESSION ) ) {
    $_SESSION = $SESSION;
  }
  $_FILES = $FILES;
  $_ENV = $ENV;
  $_SERVER = $SERVER;

}
unregister_globals ();
```

If you want to use any global variables apart from the superglobals (such as $HTTP_POST_VARS for example) you can modify the script accordingly. Keep in mind, however, that most of the ones I have not included are deprecated. The only interesting part of the added lines is the `if()` surrounding $_SESSION. This is because if no session variables are set $_SESSION is undefined. Using undefined variables isn't really a problem in PHP, but some servers (mine for example) are set up to be picky and will produce ugly warning messages.

I prefer this method to just turning off register_globals, and it can be a good patch for scripts which are written assuming register_globals are off, though it adds more code to be compiled and run for each of your scripts and it is easy to forget to use it. If you forget to unregister the global variables, you unfortunately don't get errors or anything noticeable, you just get a page that can be hacked.

Method 3 : Functions

This is by far my favourite way of avoiding the register_globals issue, and I use it for most of the pages I write now. I simply enclose all my code inside a function and then call that function. Even if the global variables exist outside the function, inside you can act as if they don't exist. Here is an example of my diary script using this method.

```
function diary () {
  if ( $_REQUEST [ 'username' ] == 'Grendels'
       && $_REQUEST [ 'password' ] == 'xxxxxxxx' ) {
    $logged_in = true;
  } else {
    show_login_form ();

  }
  ...
  if ( $logged_in == true ) {
    show_diary_entries ();
  }
}

diary ();
```

As you can see a few extra lines of code (3 to be exact) protect my code from register globals no matter how the server is set up. I keep the code indented to make it clear that it is in a function. If I come across code that is not indented in any of my files I am very suspicious. This takes a little time to get used to but if you use it regularly it becomes second nature.

Method 4 : Initialize Your Variables

For people who don't like having all their code indented and for everyone who wants to know what is in theory the "proper" way to protect yourself I have included this method, although I have found it less reliable than method 3 above. The basic idea is you initialize any variable to a safe default value before you use it. Under this system my diary login code would become.

```
$logged_in = false;
if ( $username == 'Grendels'
       && $password == 'xxxxxxxx' ) {
  $logged_in = true;
} else {
  show_login_form ();

}
```

It is the `'$logged_in = false;'` line that protects you against people setting the value themselves. On the

face of it this is a good method. However I find that all too often I add another variable somewhere and forget to initialize it properly. The code is only secure when you remember to do something, rather than being secure unless you do something stupid. In the end it is up to you which method you use, but you should choose one method and stick to it.

If You Build a Wall they Come in the Gate

Now that you have prevented hackers from sending you information in places you did not expect, you may think you have protected yourself against their attacks. If only life were that easy! If they can't send you information in places your are not expecting, the next thing hackers will try is sending the information where you are expecting it, but not sending the information you expect.

Unfortunately, we'll need to review some theory in this section, since the ways attacks can happen vary greatly between scripts, and most are quite complicated to exploit. The techniques you will learn here aren't designed to defend your site against anything specific, but to give a level of protection against a whole range of things that could happen. In general, they are not very complicated, but as with any part of your application that is important to security, you should be careful with them and test anything you write thoroughly.

Controlling What You Let in

Whenever you ask the user for information, you generally have a good idea what kind of answer you should receive. For example, if you have a text box for people to enter their age you should get back a number somewhere between about 5 and 130. Anything else is clearly someone messing around. One of the best things you can do to ensure your code is bullet-proof is to use functions to check that each value received from a user is valid. A simple function for checking something that is meant to be an age might be:

```
function is_valid_age ( $param ) {
  if ( is_numeric ( $param ) && 5 < $param
       && $param < 130 ) {
    return true;
  } else {
    return false;
  }
}
```

This is so simple that many people wonder why I bother making a function at all. The best way I can answer that is to show you two examples, one with the function and one without.

Using a function:

```
if (is_numeric ( $_POST [ 'age' ] ) && 5
       < $_POST [ 'age' ] && $_POST [ 'age' ] < 130)
{
  echo 'Hello! You are '.$_POST['age'].' years old.';
} else {
  echo 'Bad Age!';
}
```

Without a function:

```
if ( is_valid_age ( $_POST [ 'age' ] ) {
  echo 'Hello! You are '.$_POST['age'].' years old.';
} else {
  echo 'Bad Age!';
}
```

The difference in this particular case isn't very big. The significant difference is that the first example gives you a better idea as to the intent of the code. It also means if sometime in the future people live, on average, to 200 years of age, you can update all your age checks by only changing one number in your code. The difference will become more obvious as we start using regular expressions.

A Real World Example

Let's return to my diary. After showing it to a few of my friends, they asked me if I could make them copies. I decided instead that I would modify it so it could keep diary entries for any number of people. That meant I couldn't just hard-code the values for username and password anymore. I decided to use access functions to make sure the username and password values people chose couldn't do anything nasty to my computer.

I decided I would set the following restrictions on the username and password.

Username

- Must be at least 3 characters long, but no more than 15
- Must only contain uppercase and lowercase letters, numbers and underscores
- Must start with a letter

Password

- Must be at least 5 characters long, but no more than 25
- Must only contain letters, numbers, or symbols from the following list: ! £ $ % ^ & ( ) [ ] { } @

These both illustrate two important concepts in data validation. The first is the length of the value. You should always have an idea of how long or short your data is likely to be and limit the input to that. If nothing else, this keeps your application from taking up all the processor time by trying to process absurdly long

strings. The second is the concept of allowed and denied characters. In general, you should have a list of what is allowed and forbid everything else. For example, someone's name is likely to contain letters, spaces and maybe hyphens. A name field should only accept these and nothing else.

The username also shows another type of restriction you can have on your data: giving different restrictions on the first character. It is relatively simple but there is no reason not to have a set of allowed values for each part of your string. For example if you wanted the user to give you a range of values you might check the string contained a '-' that wasn't at the start or end.

So now to the actual code itself. From the amount of time it took to get here you might expect a long and complicated function. In fact, this is not the case. Anyone who has used regular expressions probably has a good idea of what it will look like. For everyone else, here it is:

```
function is_valid_username ( $param ) {
  // Check for a letter, followed by 2-14 other
  // letters numbers or underscores
  return preg_match (
        '/^[a-zA-Z][a-zA-Z0-9_]{2,14}$/', $param );
}
```

This tutorial is not here to explain in detail how regular expressions work. The only part I will highlight is the ^ and $ at the start and end of the string. These make sure the entire string must match, not just part of it. If they were not there any username containing a letter would match.

The regexp for the password is a bit more complex, as it contains characters that are normally considered by the preg_match functions to have a special meaning. To get around this, they are escaped with backslashes. The function is:

```
function is_valid_password ( $param ) {
  // Check for between 5 and 20 letters, numbers
  // or one of the following:
  //                  ! £ $ % ^ & ( ) [ ] { } @
  return preg_match (
        '/^[a-zA-Z0-9!£\$%^&\(\)\[\]{}@]{5,20}$/',
                      $param );
}
```

With a little ingenuity and a good knowledge of regular expressions, you should be able to create access functions for all your needs with little or no problem. There are only two bits of advice I will give you here. The first is to test everything. No matter how simple a function is you should check it with some values that should work and the same number that don't. You should test cases that nearly work but don't quite, like strings that are slightly too long or contain one or two invalid characters . In the same way, you should also check strings that are as long and as short as allowed and contain every allowed character.

The other tip is to document every regular expression you use. In the examples above I described in English what each regexp did just above it. Doing the same will make life much easier if you come back to your code later and need to understand what the code there is doing.

Now We Have Them, Let's Use Them!

Now let's move away from writing validation functions and move on to using them. If you are still unsure about using functions rather than just putting the checks straight into the code, go back and look at the example dealing with age and imagine a complex regular expression there instead of the simple comparisons it uses. (No, I don't consider the regexps used to check the username and password above complex.)

The most important issue in using validation functions is remembering to. Using them only half of the time is a dangerous waste. To get benefit from them you have to be consistent . I do this by checking the variable while it is in one of the superglobals (`$_POST`, `$_GET`, etc) and, if it is valid, I transfer it into another variable. I never use the superglobal variables directly in my code. If I want to check back later I just search my code for `$_POST`, `$_GET` and `$_REQUEST` and check for a corresponding 'is_valid_' function call for each one.

Another issue that can cause people problems is what to do if a value they are checking doesn't match. A lot of people send it back to the user and ask them to correct it. Although this may sound convenient, it is generally a bad idea. Data which is invalid should be discarded. In almost all cases it should not be sent back to the user, or stored in a database or even logged in an error file. There is no point in storing or saving things that could cause your code to behave unexpectedly . The user should be told it is invalid and politely asked to enter it again.

A common mistake is to only check things which the user has manually entered. A clever user can easily modify any information they send the server, so cookies, browser details and other information should be checked too. Anything which has passed through the user's computer is potentially dangerous.

Cleaning What's Left

Even if you are checking everything from the user carefully, there are some cases when you have to allow them to send characters that could be dangerous. For example, a webmail client can't refuse to send emails containing characters other than letters, numbers and spaces. The only real way to avoid problems in this sit-

uation is to have a good understanding of exactly where your data is going. This affects what is safe and what isn't. For example, things going to a shell containing ';' and '|' are dangerous, as they could allow someone to run commands on your server, but they are not generally very dangerous to store in files.

To give you an idea of the kinds of things that can be dangerous, I'll take you through some common examples. I make no claim that the lists of dangerous characters I give here are complete, nor is this a replacement for Reading The Fine Manual. I'll also look at ways of allowing these characters while preventing them from doing harm.

HTML

The most common format to which data from your users will be transformed, sooner or later, is HTML. Whether this is your public website, or some private back end, allowing arbitrary HTML written by a user to be displayed is a security problem. The danger can range from showing a fake form asking users to reauthenticate and getting their passwords to using scripts which can silently perform any action the current user can do, to completely changing the look of your site.

Removing the threat of HTML in PHP is very easy. PHP has two built in functions, '`strip_tags()`' and '`htmlspecialchars()`'. The first of these will remove any HTML tags from a string. The second will convert unsafe HTML characters like '<' and '>' to their encoded counterparts (< and > respectively). Personally I prefer the '`htmlspecialchars()`', as it allows users to enter text contained within angle brackets without it getting removed, but if your users regularly try to use HTML, strip_tags will give a neater looking output.

I do not recommend trying to identify certain "safe" HTML tags and allowing them while disallowing others. There are hundreds of ways of including scripts in a page, so unless you are an expert in how all common browsers render HTML (I know I'm not), you are likely to make mistakes. Hotmail and Yahoo keep getting it wrong, so you are likely to as well.

You should be especially careful of any text inside HTML tags, e.g. allowing custom image URIs. This further multiplies the number of ways scripts can be run. If you must do this, I recommend that you remain very strict about the format of all of your data.

SQL

Whenever general web application security is mentioned, SQL injection is never far away. SQL injection is when an attacker uses malicious data to change an SQL query, making it do something other than intended. For example supposing you had a query like

```
sql_query ( "SELECT * FROM members WHERE username='$username' AND password='$password'" );
```

This would normally select all details about the current user from the database (checking their password at the same time). However, with a bit of imagination it can be made to do something more. Supposing `$username` is equal to `'; DELETE FROM members; —` . When the query is constructed by filling in the variables it becomes:

```
SELECT * FROM members WHERE username=''; DELETE FROM members; – ' AND password='pass'
```

As you can see the query has become two queries, delimited by the semicolon (;), and will neatly wipe out your member list. The '—' at the end tells the SQL parser everything following it is a comment and prevents any parse errors. In SQL possibly dangerous characters are ", ' and \, assuming all your values are inside quotes.

In spite of the potential severity of SQL injection, usually while programming in PHP you don't need to worry about it as often. The first reason for this is the PHP setting called magic_quotes_gpc. This automatically escapes any dangerous data coming from the user. For example `'; DELETE FROM members; —` would become `\'; DELETE FROM members; —` and would prevent the sql engine thinking it had reached the end of the string. This does not mean you can completely forget about SQL injection. Data which does not come from the GPC variables (e.g. something selected from the database itself) is not protected. For this you need to use an escaping function.

The built-in escaping function in PHP is '`addslashes()`'. This should be safe enough. However, some database interfaces also provide their own functions, such as '`mysql_escape_string()`'. If the extension your database uses has a function like this I suggest you use it. This means your code is more likely to stay correct if that database engine is changed in some strange way.

The other reason it is not often a problem only applies to the MySQL database engine, but since the majority of PHP powered sites on the web use MySQL I feel justified in mentioning it. MySQL only allows one query per call to mysql_query, so in fact it is impossible to add another query adding or deleting from the database. It is still possible, however, to modify the query that is run, and MySQL version 4.0 offers many more features, so you should still be careful.

Files

Files are in some ways the easiest place to store data safely. In general you can put anything you like in a file without fear of the underlying filesystem having prob-

lems. One problem with files comes if you are not writing the program that reads them. If this is the case you should make sure you fully understand the file format expected by the program that reads it.

Remember that programs that might read these files include your web server. If you are using files for storing information make sure they are in folders unavailable from the client. If that is not an option, try using .htaccess files in apache, or the equivalent configuration settings in your webserver to restrict access. Keep in mind, though, that keeping the files in an inaccessible folder is a better option.

The luxury of assuming the safeness of the contents of files does not carry over to the file name. Again if at all possible you should not use user-supplied data in filenames. If you have to, you should reject any names containing '\', '/', ' ' or '.'.

system() and exec()

If you are passing data to another program by using functions like '**system()**' or '**exec()**' you must be very careful. Again you should be as strict as possible with what you let in. My suggestion is to surround any arguments from the user in single quotes. For example instead of using

```
exec ( "./log_access $user_var" );
```

do

```
exec ( "./log access '$user_var'" );
```

This means most shell escape characters like ';' and '|', which allow other programs to be run will just be treated as normal characters. A nasty attacker could break out of this by adding another single quote somewhere in their data, so you have to escape all of those characters with '\s'. All '\s' should also be escaped. This can be done simply with '**str_replace()**'.

```
$user_var = str_replace ("\\", "\\\\", $user_var);
$user_var = str_replace ("'", "\\'", $user_var);
```

The backslashes have all been doubled to make PHP treat them as normal characters.

Better still, whenever possible, you should not allow '\', or '"anywhere in the string, and preferably not ';' (new command) '|' (pipe to another command), or the redirection operators '>' and '<' either. You should study the documentation for your environment for any other characters with special meaning and make sure they are excluded too.

Trying not to GET the POST

Another issue that many PHP scripters overlooked while `register_globals` was widely used was the distinction between GET and POST data. Many people would also say there is, in terms of security, essentially no difference between them. An attacker can fake either of them easily and neither offers much protection for the data in transit. They would also say that any problems with proxy servers caching GET data should be avoided by setting the type of form used and checking that the script doesn't add anything.

The HTTP spec says that anything which affects data beyond the current request must be sent as a POST. No problem with this, we can just make sure our forms use POST, alleviating the need to code our scripts to check that data was sent using a POST operation. In spite of what is commonly believed however, quite a lot of the time an attacker can trick a user into sending GET data but not POST data. The best way to demonstrate this is with an example.

Image you run a set of forums on the Internet that use cookies. You decide you want to allow users to include images in their posts. This all works fine until one day when someone comes along and enters an image with the URI http://www.my-cool-boards.com/admin.PHP?delete_all_posts=true. Most people will just see a little cross to say the location given isn't an image. When the admin comes along and looks however it is a bit more serious. He sees the little cross, but then when he goes back to the main page he finds all the posts on the forums have been deleted. As the request was made from his browser session the site thought it was him asking to do it and happily deleted everything!

By far the simplest and possibly the safest way to avoid problems caused by register_globals is simply to turn it off.

If his script had checked to insure that the data was POSTed nothing would have happened. When loading up that page it would have seen the data was GETed and ignored the request. Several of you are now probably thinking: Well the forums shouldn't use cookies for authentication. It is true that cookies on their own are lacking as a way of identifying people and in general you should combine them with a method like URL encoded values. You should, however, always be aware that GET data is just slightly easier for attackers to fake

than POST data. If it should always be POSTed, it is good practice to check.

Hackers will try sending the information where you're expecting it, but not sending the information you expect.

So how do you make sure your apps only respond to POST requests? The first method is easy, simply use the `$_POST` superglobal instead of `$_REQUEST` to access your variables. In any situation where you have a form being filled in or a button clicked to determine what should be done , accessing the value using `$_POST` is recommended. There is another way to perform this check which some may prefer, which involves using a simple `is_posted()` check that will work with all forms. In fact, it is your only option if you are not passing any values to the page.

To use this method, just look at the `$_SERVER['REQUEST_METHOD']` variable. If the request to your page has been posted it will be equal to 'post', although not necesarily lower case. The is_posted() function I mentioned earlier could be coded simply as

```
function is_posted () {
  return strtolower (
        $_SERVER [ 'REQUEST_METHOD' ] ) == 'post';
}
```

A simple extension on this idea would be a `check_post_request()` function that printed out an error message, then terminated the script if the form was not posted.

Just When You Thought it Was Safe...

All the items mentioned in this article are important and necessary if you want to create a safe application. I have tried to focus on issues that can be brought about by an attacker relatively easily, just by giving your application information it does not expect, or in a way it does not expect. This is by no means a conclusive look, however, and it should be noted that you can follow all of these guidelines perfectly and still have an insecure site. Absolute 100% security is impossible, but if you want to make your sites as hack-resistent as possible you should not stop here.

There are lots of resources available on the Internet, and plenty of people out there willing to answer your questions. Read everything you can that deals with security, and slowly you'll learn where all the holes are and how to plug them (well, most of them anyway).

php|a

Theo Spears is a college student based in the UK. He learned PHP in 2000 to avoid having to use perl for a web project and has successfully been using it to avoid perl ever since. Theo now spends most of his coding time working on systems to make it easier to write secure and versatile applications. You can reach him at postmaster@terrarium.f9.co.uk

www.cyberbite.com
CYBERBITE
WEB HOSTING
Revolving around YOUR business
Designed for PHP Programmers
Virtual Private Servers & Dedicated Servers
Reliable Internet hosting solutions
Guaranteed 99.95% uptime

Reviewed For You

ionCube PHP Accelerator by ionCube Ltd.

Price: N/A (free)

Pop quiz, hotshot: your PHP-based website has suddenly risen in popularity. Your traffic has increased by 50%. Your manager is breathing down your neck. What do you do?

From a performance perspective, the problem (and, in some cases, the advantage) of a language like PHP is the fact that scripts based on it have to be read, interpreted and executed by the PHP engine every time they are run. By contrast, compiled languages like C produce output that is in native binary code and can be executed directly by the operating system (but is not portable).

Although for the most part the overhead caused by parsing the scripts is negligible on a server that does not have to support a significant amount of traffic, for a very popular website it can be a big problem. This is because, on a large scale of tens of thousands of page impressions, the time that it takes for the PHP engine to transform a source file from text into execution data that it can process (also known as bytecode) adds up pretty quickly and can become a significant performance impairment.

Enter the Cache

If your site's popularity grows by some disproportionate magnitude overnight, the first impulse is always to add more hardware. There are, however, a number of problems with adopting this approach. First of all, hardware is expensive—at least in relation to open-source software. Second, it takes time to set it up; you have to order the parts, install all of the software and then somehow set up your server farm so that your traffic is balanced among multiple servers. Third, it is very expensive to maintain. Each new server means a new machine that must be patched, monitored, cleaned and that could be yet another door through which a hacker could gain access to your data.

A software solution—preferably one that is easy to implement—should therefore be the first choice when you're at odds with your site's performance. Caching engines (CEs) may be exactly what you need.

A CE works by "capturing" your script once it has been converted to bytecode the first time that it is executed. The next time the same script is requested, the caching engine simply forces PHP to run the bytecode version without going through the whole parsing process once again and, in so doing, drastically reduces the overhead involved with fulfilling a request made by a client for the script.

Some other scripting platforms, like Microsoft's ASP, provide this mechanism as part of their standard distributions. Some others, like Java, solve it by actually compiling the code directly into bytecode format. In PHP, however, this optimization is not included in the standard package and must be performed by an external module.

This is, essentially, a "good thing", since it makes it possible for different caching engines to exist and to compete for the best performance possible. It also means that, if your site is having performance issues and you never considered the use of a CE, you've just won the lottery and the right caching strategy can make the difference between running around like a chicken with its head cut off and running around with your head cut off like a chicken (I'll let you figure out which one is best).

ionCube PHP Accelerator

There are, in the true spirit of capitalist competition, a number of caching engines for PHP available on the market. In this review, we will examine the ionCube PHP Accelerator, or PHPA (not to be confused with php|a!). Produced by U.K.-based ionCube Ltd., PHPA is a very mature application that has gone through several releases, is well tested and very well supported (more about that later).

PHPA is available free of charge, but it is not an open-source product (although its website hints at the fact that the source code might be released in the future). Luckily, there are binaries available for most platforms, including Linux, Solaris and Free/NetBSD. There is no support for Windows or MacOS at this time.

Although PHPA comes with no formal setup application, installation is truly a breeze—and, at least on Linux, the package is tiny (only about 55kB). All you have to do is copy the accelerator's library and add a single line to your php.ini file. PHPA configures itself as a Zend extension, which gives it complete access to the PHP engine.

Caching Strategies

All the caching engines available for PHP support two main caching strategies, one based on temporary files and the other on shared memory. In the first case, the CE stores the bytecode version of a script in a temporary file and then reloads it every time that script is executed. Although this is a big improvement over having to recompile the script every time, it's not as efficient as its shared memory (SHM) counterpart, in which a single cache object is created and shared among several instances of PHP.

The reason why all engines implement both schemes is that SHM is not available on all platforms and, therefore, temporary files must be used in some cases even though they do not provide the best performance.

Performance

So, how much of a difference does having PHPA make? Simply put—a lot.

We performed a series of tests to verify how well PHPA does in relation to "plain" PHP and two other very popular acceleration engines, the Advanced PHP Cache (APC) and the Zend Performance Studio. APC is a true open-source CE, while Zend Performance Studio is a commercial product published by Zend Technologies that also includes a content caching engine (that is, a CE that caches a page output rather than its compiled source code), which we turned off for our test, so that we could (hopefully) compare apples to apples.

We ran a series of tests using three scripts from the php|a websites. Index.php, our main page, is a light database-driven script that performs mostly simple queries and text output with little logic behind them. Subscribe.php, on the other hand, is a very complex script with more than one thousand lines of code. Finally, Advertise.php, our advertiser information page, features no database access at all and contains very little PHP code. The idea behind these choices is to show how well a CE performs in three different scenarios of varying complexity.

Listing 1 shows the code used to perform these tests. As you can see, we essentially retrieve a particular web page from our test server one hundred times. Our Apache/PHP 4.2.2 server was reset with a hard reboot before each run (to ensure a fresh start every time) and the test script was run through the CGI version of PHP with all caching turned off, to ensure that the CE would not affect its performance and skew the results.

In our measurements, we looked at the "time to first response", the "time to second response" and the average execution time over 100 responses. The first value tells us how efficient the compilation process is, since the first time we run a script the CE has to parse it, optimize it and store it in its cache. The time to second response tells us how efficient the script becomes right after it has been cached. Finally, the average over 100 responses provides a good overview of the performance improvement provided by the CE. It should be noted that, in a real-life environment, the number that

Listing 1

```
<?php

function getmt()
{
    $a = explode (' ', microtime());
    return (double) $a[0] + $a[1];
}

$file = "http://127.0.0.1/advertise.php";

for ($i = 0; $i < 100; $i++)
{
    $start = getmt();
    file ($file);
    echo (getmt() - $start) . "\n";
}


?>
```

Figure 1

	PHPA			PHP (no cache)			Difference		
	Time to 1st	Time to 2nd	Avg/100 times	Time to 1st	Time to 2nd	Avg/100 times	Time to 1st	Time to 2nd	Avg/100 times
index.php	0.0209	0.0135	0.0111	0.0190	0.0177	0.0187	-9.72%	23.50%	40.84%
subscribe.php	0.0698	0.0351	0.0213	0.0598	0.0538	0.0457	-16.66%	34.71%	53.48%
advertise.php	0.0090	0.0075	0.0075	0.0175	0.0170	0.0172	48.54%	56.01%	56.41%

Figure 2

	PHPA			APC			Difference		
	Time to 1st	Time to 2nd	Avg/100 times	Time to 1st	Time to 2nd	Avg/100 times	Time to 1st	Time to 2nd	Avg/100 times
index.php	0.0209	0.0135	0.0111	0.0261	0.0169	0.0169	19.97%	19.98%	34.52%
subscribe.php	0.0698	0.0351	0.0213	0.0632	0.0427	0.0395	-10.51%	17.69%	46.23%
advertise.php	0.0090	0.0075	0.0075	0.0372	0.0374	0.0406	75.83%	80.05%	81.57%

Figure 3

	PHPA			ZPS			Difference		
	Time to 1st	Time to 2nd	Avg/100 times	Time to 1st	Time to 2nd	Avg/100 times	Time to 1st	Time to 2nd	Avg/100 times
index.php	0.0209	0.0135	0.0111	0.0198	0.0134	0.0110	-5.45%	-1.04%	-0.73%
subscribe.php	0.0698	0.0351	0.0213	0.0502	0.0233	0.0204	-39.15%	-50.75%	-4.32%
advertise.php	0.0090	0.0075	0.0075	0.0076	0.0069	0.0072	-18.29%	-8.12%	-3.89%

really counts is the averaged response time. The first- and second-response measurements are just a way to figure out how good the CE's parsing system is and, since they are relevant only the first time the script is executed, would not affect the performance of a web application in a production environment.

We started by testing PHPA against plain PHP. As you can see from Figure 1, the overall performance improvement provided by PHPA is very significant, even if you keep in mind that these tests were run in "optimal" conditions, that is, with only one client at a time requesting pages in a serialized fashion. In a live environment, with several clients hitting a script at the same time, PHPA would provide the additional benefit of reducing hard disk usage, thus increasing the performance even more. Not surprisingly, the biggest improvement is in subscribe.php, which is also the most complex script. You will also notice that PHPA's time to first response is slower in the first two cases. This is due to the fact that the CE has to cache the script—something that the PHP engine doesn't normally have to do.

Comparing PHPA with its competitors provided us with some interesting results. As you can see in Figure 2, APC is significantly less likely to make a significant difference in the performance of PHP than PHPA—so much the pity, since it is open-source and we all could have learned from it. Unfortunately, APC has also suffered from a lack of updates and active development for quite a while now.

Figure 3 shows our comparison with the Zend Performance Suite (which we reviewed in last month's issue). Again, keep in mind that we tested the ZPS with the content caching engine turned off (and, as you may remember, that made quite a difference in our previous tests). Still, it performed slightly better than PHPA, although not significantly so.

Customer Support

With a product that is so tied with the core functionality of PHP, customer support is a feature you don't want to miss on. Although the words "review" and "magazine" usually have a magical effect on getting companies to return requests for information quickly, we were simply amazed at how dedicated the PHPA team is to their product—suffice it to say that our requests for assistance were answered in less than ten minutes on a Sunday afternoon. Anyone who's ever been on the phone with Microsoft Technical Support Services should have a fair idea of how good this response time is!

The Bottom Line

Our conclusion is that PHPA is a product worth having as part of your web installation. Although it's not as complete as the ZPS, which also features a complete installation package, as well as built-in content caching and output compression, PHPA comes with an unbeatable price and a customer support that many companies still can't provide.

php|a

The OPEN SOURCE WEEKEND

A Weekend of Activities and Information on Open Source Software

Ottawa, January 25 & 26, 2003

Linux-Fests

Saturday, January 25, 11AM - 5PM

Carleton University: Porter Hall, in the Unicentre
University of Ottawa: SITE building

including:

- presentations by IBM, Sun, and local open source experts
- installs of the Linux Operating System
- seminars, tutorials and beginners sessions
- tradeshow displays

The Business of Open Source Software

Sunday, January 26, 11AM - 5PM

with keynote speaker

MARTIN FINK, author of
The Business and Economics of Linux and Open Source

and a panel discussion with open source experts from government and industry:

Danese Cooper	Open Source advocate, Sun Microsystems
Eid Eid	President and CEO, OEone Corporation
Jim Elliott	Linux advocate, IBM Canada
Joseph Potvin	PWGSC, Government of Canada

Location: Clark Room, RA Centre

Please visit **www.osw.ca** for more information.

We thank our sponsors:

php|architect

IBM

Sun microsystems

ROARING Penguin Software Inc.

OEone

FEATURES

Writing a Web-based PDF Viewer

By Marco Tabini

There is nothing as annoying as having to download a 10 MB PDF viewer to be able to read a two-page document. With a bit of help from a few open-source friends, it is actually possible to write a web-based PDF viewer that requires no plug-ins or external applications on the client side.

Quite a large number of websites use PDF files as a way to distribute some of their information. Aside from companies that, like php|a, make extensive use of this format, many others use it for small documents, such as forms, prospecti, brochures and so on. Clearly, they find PDF attractive because of its typographical accuracy and the compactness of its files.

There is, of course, one big assumption behind all this—that the users have a PDF viewer. If they don't—and many people still don't—no matter how small your document is, they will still have to download a viewer, and Adobe Acrobat is (in its simplest form) an 8 MB monster. As it turns out, while you think that you're providing a valuable service to your users by offering them a slim and elegant PDF document, you run the risk of alienating them because not everyone wants to sit around while his or her 56kbps modem downloads a huge program to read two pages of information!

A Simple Solution

There is a simple solution to this problem. If you think of PDF files (once rendered on a screen or on a sheet of paper) as very detailed images, it's not completely inconceivable to think that you could, indeed, serve them as such from your website. It is therefore possible to create a simple, yet complete (in terms of detail and typographical fidelity) web-based PDF viewer that requires no software downloads or installations, and no plug-ins, with the added bonus that it will work on any browser capable of displaying images.

Clearly, an image depicting a PDF document's page has to provide a certain amount of detail, so that it will look pleasant and professional to the user. As such, you can expect file sizes that would normally be considered too large for a web application—say in the one hundred to two hundred kilobytes range—but that are very small when you compare them with the size of a PDF viewer.

Still, there is one major issue that you should take into consideration, and that's bandwidth usage. In fact, while most users will be downloading a viewer directly from the website of the company that produces it (for example, Acrobat is usually downloaded directly from Adobe's website), if you want to serve the file's pages as images, you will have to do it directly from your servers. Depending on the amount of traffic that you serve, this can mean a significant increase in the amount of bandwidth that your site uses, which, in turn, can result in additional expenses for your company.

Enter Ghostscript

As I mentioned in my PDF conversion article pub-

REQUIREMENTS

PHP Version: **4.0 and above**
O/S: **Any**
Additional Software: **Ghostscript 5.0 and above, ImageMagick**

lished by php|a last month, Ghostscript is a very sophisticated PostScript interpreter that provides a wide array of functions. One of these is the ability to create rasterized image versions of PostScript documents. This should not come as a particular suprise, since Ghostscript is very often used as the underlying software for viewing PostScript files on a computer's monitor, and this obviously requires that it be able to generate bitmaps that can be visualized on-screen. From there, it's a short jump to being able to save these bitmaps in one of several standard formats, such as JPEG and PNG. In fact, a quick look at Ghostscript's functionality reveals that it supports several variations of either of these standard formats, as shown in Figure 1.

A problem that is immediately obvious, however, is that image formats can usually only contain one frame at a time in a manageable way. GIF, for example, supports animations, but it is neither open-source nor easily manageable, since there's no way to control how each frame of an animated image is displayed. MNG (pronounced "Ming") is an open-source alternative to GIF, but it is not widely supported by the most popular web browsers and suffers from the same manageability problems. What's more, at high resolutions our PDF images are likely to become very large, and storing them in a single file would pose additional—and useless—strain on the application's bandwidth usage.

Figure 1 - PNG and JPEG Bitmap devices

png16	16-colour PNG (4 bpp)
png256	256-colour PNG (8 bpp)
png16m	True-colour PNG (24 bpp)
jpeg	True-colour JPEG
jpeggray	Gray-scale JPEG

Luckily, Ghostscript makes it possible to easily overcome this issue by providing a file naming mechanism that can be used to separate a PostScript file's converted version in individual files, one for each page. All that's needed is the addition of the %d (or %ld) parameters to the file name. For example:

```
gs -dNOPAUSE -dBATCH -sDEVICE=png256
  -sOutputFile=test%d.png test.pdf
```

will create a set of files named test1.png, test2.png, and so on, depending on the number of pages contained in test.pdf.

Handling Zoom and Making Images Prettier

If you try to manually convert a PDF file into a series of images using Ghostscript, you'll find that the output quality is not particularly good: the fonts, in particular appear jagged at the edges and not as smooth as we've come to expect from the commercial PDF viewers. This is because the Ghostscript rendererer was not really designed for on-screen output, but rather for printer devices and, therefore, there are no provisions in it for "anti-aliasing", the technique used to smooth the appearance of fonts on-screen.

Another problem in need of a solution is how to handle zooming. Unlike PDF files, pictures cannot be zoomed in and out with a browser and, even if they could, their quality would suffer, because their resolution is fixed at the time of creation. A possible solution would be letting Ghostscript generate a different set of images at different resolutions by using the -r command-line switch. For example, "100%" zoom-level would correspond to 72 dpi (your typical screen resolution), which, for a normal letter-size page would give us a width of 8x72 = 576 pixels and a height of 756 pixels. Similarly, 200% zoom would correspond to 144 dpi, or an image 1,152 pixels wide by 1,512 pixels high.

While this is the solution that I have ultimately adopted for my PDF viewer, I wasn't very satisfied with its initial implementation, which relied exclusively on Ghostscript. For one thing, PDF rendering can be very slow, depending on the complexity of the document being converted. Wasting an inordinate amount of time in this process didn't seem like a good choice to me, since uers tend grow annoyed quickly these days. Additionally, this still didn't get rid of the image quality problems, s the fonts still appeared jagged no matter the output resolution.

To improve my PDF viewer implementation, I turned to another freely available (and open-source) product called ImageMagick, which was developed by John Cristy and whose home page can be found on the Web at http://freealter.org/doc_distrib/ImageMagick-5.1.1/. ImageMagick is a set of tools designed for the manipulation of images with an impressive line-up of features, including format conversion, resizing, resampling, drawing, and so on.

Although ImageMagick provides several command-line interfaces, we're only interested in a tool called mogrify. Among this tool's many useful features is the ability to resample an image using a bilinear algorithm. Invoking mogrify to resample a page to a certain resolution requires the use of just a handful of parameters:

```
mogrify -scale width filename
```

Based on what we have looked at so far, our page-display strategy will work as follows:

1) Fetch PDF file from user
2) Convert PDF to PNG-256 format using Ghostscript at high resolution (100-300 dpi)
3) Whenever asked to visualize a page, resample it to the appropriate zoom level using mogrify

As you can see, the process is rather simple, but it poses a few challenges. First of all, we need more control over the PDF file—how do we know that the user won't upload a file that is 200 pages long, thus causing significant strain on our servers? Moreover, if the user asks to view the same page at the same zoom level, we should provide some sort of caching mechanism to prevent wasting time and resources by mogrifying again. Finally, we should prevent the user from copying the images directly from our web server, so that our PDF viewer doesn't become the "free PDF-to-PNG converter of choice" for the web!

Limiting the Page Count

There are a number of different ways to determine how many pages a PDF document has. I have chosen to use a simple program called **pdfinfo**, which was developed by Derek D. Noonburg (who also happens to be the author of the free PDF viewer xpdf) and is freely available from http://www.opengroup.org/infosrv/PDF/xpdf/.

Pdfinfo is highly portable and can be run on a number of platforms, including several flavours of UNIX and Windows. In addition, it is tiny and self-contained—which also means that it's fast and doesn't use too many system resources when it runs. Finally, it fully supports encrypted PDFs, thus making the upgrade of our viewer to include that functionality quick and easy.

The interface to **pdfinfo** is rather simple. In fact, for our purposes we only need to pass to it the name of the PDF file that it must read:

```
pdfinfo pdffile
```

A call to **pdfinfo** returns a list of parameters taken from the PDF file—Figure 2 shows an example—including its page count. All we need to do is parse the results using the simple script shown in Listing 1, and we will be able to use the expression `$args['Pages']` to determine whether the PDF is too big for us to process.

Listing 1

```
if (!exec ("pdfinfo {$_REQUEST['userfile']['tmp_name']}", $args))
    die ('Error converting PDF file [possibly encrypted?]');

foreach ($args as $v)
{
    $data = explode (":", $v);
    $args2[trim ($data[0])] = trim ($data[1]);
}
```

File Organization and Caching

To make it easy to manage the many files that are generated by the PDF viewer, I've used a simple file naming convention. When the PDF file is initially converted over to a high-resolution set of images, we start by generating a temporary file name and passing it to Ghostscript using the %ld parameter. As mentioned earlier, this effectively creates a set of files that end with the number of the page they contain.

Figure 2 - Typical pdfinfo output

```
Producer:      GNU Ghostscript 6.52
Tagged:        no
Pages:         11
Encrypted:     no
Page size:     595 x 842 pts (A4)
File size:     77274 bytes
Optimized:     no
PDF version:   1.2
```

Next, a different file has to be created for each of the various zoom levels. Since the application can handle several different levels, it is not a good idea to generate all the pages right away, as that can potentially take a significant amount of time. Therefore, the application generates each file using a just-in-time approach—that is, it runs mogrify whenever the user requests a particular zoom level of a specific page.Bear in mind that this approach is not very performance-conscious. One reason for this is because as the user flips through the pages of the PDF file he or she is likely to load the same page twice, resulting in mogrify being executed more than once for a particular page/zoom level combination.

This problem can be solved by simply adopting a properly devised naming convention for the zoomed-in image files as well. In our case, I simply append the suffix 's{zoom-level}' to the name of each file once it's created and allow it to exist past the execution time of the script that shows the zoomed-in image to the user. This way, whenever that script is executed, it verifies whether a file matching the proper naming convention for the current file, page number and zoom level exists, and in that case outputs its contents without invoking mogrify again.

Clearly, this approach tends to clutter up your temporary directories. However, you can easily solve

this problem by establishing a timeout period and creating a scheduled script that deletes stale files according to your needs.

Preventing Image Theft

Tweaking our code so that users cannot save our images to their hard drive is tricky—anything that affects the usability of the application under any but the most odd circumstances is a definite no-no.

The easiest way to ensure the safety of the images generated by the application, therefore, is to prevent their indiscriminate usage, rather than trying to cover all the possible scenarios in which a violation could occur. In fact, just ensuring that an automated program won't be able to run a hundred PDF files through the viewer and collect the results should be enough—I doubt that a sane person would bother to do so by hand.

The technique that I have chosen to use consists of examining the contents of the `HTTP_REFERER` parameter that is passed by the web server to PHP in the `$_SERVER` superglobal array. This parameter contains the address of the last page that the user's browser visited prior to attempting to retrieve the current one. Under normal circumstances, when an image is being downloaded it will contain a page that resides in the current domain—that would be the page that contains the <img> tag which causes the image to be shown. Therefore, if the contents of the referer string do not match the domain in which the image generation script resides, we simply interrupt the execution and output an error.

Of course, there are a number of limitations in this mechanism. For starters some older browsers do not support the referer mechanism. These are, however, very rare and represent a tiny percentage of the Internet userbase. Similarly, some "privacy enhance-

Listing 2

```
1  <?php
2 if (!isset ($_FILES['userfile']) || $_FILES['userfile']['error'] || !is_uploaded_file ($_FILES[
'userfile']['tmp_name']))
3     die ("File upload error. Please try again.");
4
5 session_start();
6
7 if (!exec ("pdfinfo {$_REQUEST['userfile']['tmp_name']}", $args))
8     die ('Error converting PDF file [possibly encrypted?]');
9
10 foreach ($args as $v)
11 {
12     $data = explode (":", $v);
13     $args2[trim ($data[0])] = trim ($data[1]);
14 }
15
16 if ($args2['Pages'] > 10)
17     die ('This converted only supports a maximum of 2 pages');
18
19 $_SESSION['args'] = $args2;
20 $_SESSION['max_zoom_level'] = 4;     // Maximum zoom level
21
22 $tempfilename = tempnam ($_SERVER['DOCUMENT_ROOT'] . '/tmp', 'pdf');
23 $tempfilename2 = tempnam ($_SERVER['DOCUMENT_ROOT'] . '/tmp', 'pdf');
24
25 copy ($_FILES['userfile']['tmp_name'], $tempfilename2);
26
27 // Prepare high-res files
28
29 shell_exec ("gs -dNOPAUSE -dSAFER -sDEVICE=png16m -r200 -sOutputFile=$tempfilename%ld.png -
dBATCH {$_FILES['userfile']['tmp_name']}");
30
31 $_SESSION['imgid'] = $tempfilename;
32 $_SESSION['docid'] = $tempfilename2;
33
34 ?>
35 <html>
36 <frameset cols="100,1*">
37 <frameset rows="1*, 100">
38     <frame src="header.php">
39     <frame src="footer.php">
40 </frameset>
41 <frame name=downtarget src="view.php?p=1&z1=1">
42 </frameset>
43 </html>
44
```

ment" systems prevent the browser from passing the referer along to the web server—those users will not be able to use the viewer at all, and it's up to you to decide whether you are willing to forego support for them in favour of at least some level of protection against image theft. Finally, the referer data is, in the end, user input and, as such, it can be faked in a relatively easy way, particularly by a program.

Still, even with all its limitations, this method provides, in my opinion, a good balance between security and simplicity and, although there are certainly other techniques that can be considered, it should be good enough for the needs of the average website.

On to the Code

The most complex part of the system's implementation is really organizing the viewer's main interface, since we need to provide a way for it to work seamlessly from the user's point of view.

In my case, I've decided to use a frameset, since most of the interface doesn't really change from page view to page view. As you can see in Figure 3, the screen is divided into three different sections. The bottom left corner contains general information—the output of the **pdfinfo** call—about the PDF file that is being viewed, while the top left part of the screen provides a simple interface to jump to a page or change the current zoom level. Finallly, the current page is displayed in the large area on the right.

As you can see in Listing 2, the main interface layout is determined by the upload.php script, which also performs the initial checks on the PDF file and invokes Ghostscript to generate the high-resolution rendition of the page images. The key parameter to keep in consideration here is -r, which is used to determine the resolution at which Ghostscript outputs its images. Giving this parameter higher values (I use 200 dpi in my example and that's already very high) will result in better results in term of quality, but slower image processing and bigger image files.

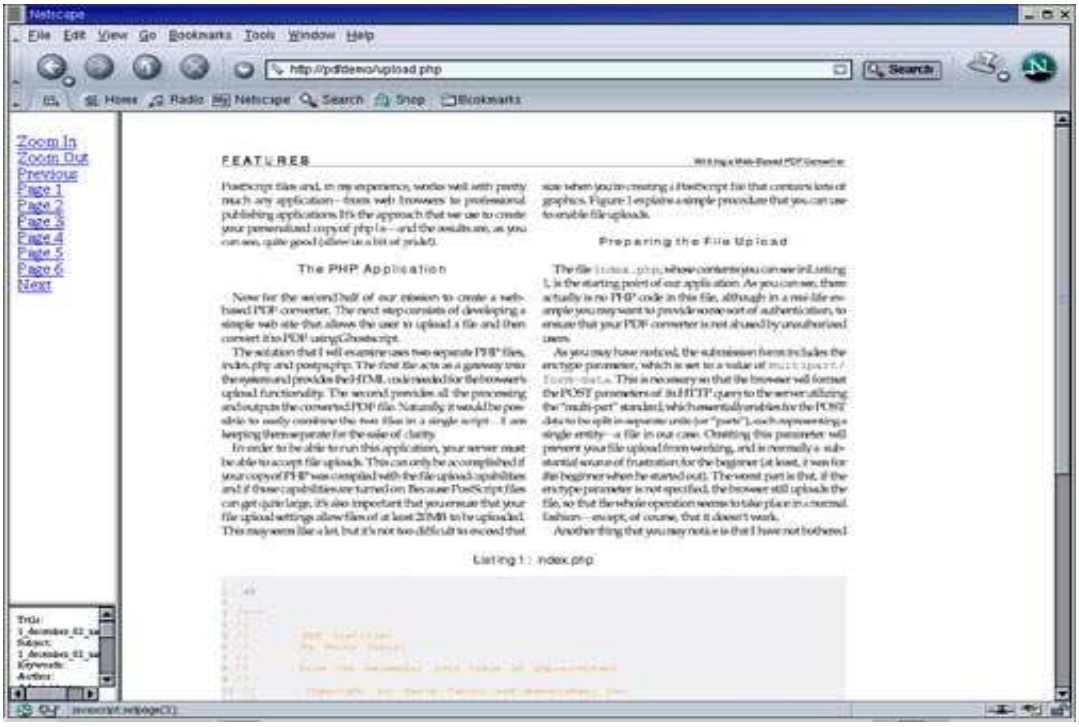
Upload.php receives as its input a file from the user, which is uploaded through the index.php page. For a more detailed discussion about file uploads in PHP, you can refer to my article on PDF conversion in the December 2002 issue of php|a.

The use of sessions in our application allows the three frames that comprise the main interface to share information about the image file names and PDF parameters without passing them explicitly through the browser, thus protecting the code from unwanted and potentially dangerous outside interference.

Listing 3

```
<?php
session_start();

$compare_string1 = 'http[s]?://' . $_SERVER['HTTP_HOST'] . '/header.php';
$compare_string2 = 'http[s]?://' . $_SERVER['HTTP_HOST'] . '/upload.php';

$page_id = $_GET['p'];
$zoom_level = $_GET['z1'];

if ((!is_numeric ($page_id) || $page_id < 1 || $page_id > $_SESSION['args']['Pages']) ||
    (!is_numeric ($zoom_level) || $zoom_level < 1 || $zoom_level > $_SESSION['max_zoom_level'])
)
    die ('Invalid access.');

$file_name = $_SESSION['imgid'] . $page_id . 's' . $zoom_level . '.png';

if (ereg ($compare_string1, substr ($_SERVER['HTTP_REFERER'], 0, strlen ($compare_string1))) |
| ereg ($compare_string2, substr ($_SERVER['HTTP_REFERER'], 0, strlen ($compare_string2))))
{
    if (!file_exists ($file_name))
    {
        copy ($_SESSION['imgid'] . $page_id . '.png', $file_name);
        shell_exec ('mogrify -scale ' . ($zoom_level * 250) . ' ' . $file_name);
    }

    header ("Content-type: image/png");
    header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");    // Date in the past
    header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
                                // always modified
    header("Cache-Control: no-store, no-cache, must-revalidate");  // HTTP/1.1
    header("Cache-Control: post-check=0, pre-check=0", false);
    header("Pragma: no-cache");                          // HTTP/1.0
    readfile ($_SESSION['imgid'] . $page_id . 's' . $zoom_level . '.png');
}
else
    die ("Invalid Access. Please try again...<p>");
?>
```

Figure 3 - PDF Viewer Interface



Naturally, under normal circumstances, sessions are still vulnerable to "hijacking"—that is, a malicious user could, in theory, intercept the browser's transmission to the server, capturing the session ID generated by our scripts as a result. That ID could then be used by the would-be hacker to impersonate the user and gain access to the PDF file he or she was reading. However, for the typical usage of this application, the level of security offered by sessions should be good enough—and you can always provide further protection by placing your pages under HTTPS.

Never Trust User Input

Since as much data as possible is managed through sessions, the only bits of input that the application receives directly from the user is the input PDF file, which is examined carefully by the upload.php script, and the page numbers and zoom levels, which are looked at in the view.php script (Listing 3). As you can see, the script ensures that these parameters are numeric in nature and that they do not exceed proper values (if they do, an error is returned). Line 16 provides the mechanism to determine whether the user is coming from within the script's domain or not, in which case the script dies with an error message.

View.php is also used to generate (or retrieve) a zoomed-in image. The `$file_name` variable is built using a combination of session and properly filtered GET variables in accordance with the naming convention that I mentioned above. The ImageMagick mogrify command is only executed if a particular image file is not already present.

A Final Word for Windows Users

A few Windows users wrote me in reference to my article on PDF conversion in the December 2002 issue of php|a to tell me that they experienced some problems running Ghostscript on their platform. After a bit of mutual debugging, we determined a couple of valuable lessons:

- The Windows version of Ghostscript is called gsw32.exe and not just gs.

- You need to specify the full path to Ghostscript, as PHP does not include it in its search path. This is true even if you can execute gsw32 directly from the command line without providing any path information.

This last note applies, of course, to ImageMagick as well.

php|a

Marco Tabini is co-editor of php|architect. He spends most of his time lurking around on the PHP mailing lists and on the php|a website doing his best to confuse fellow programmers who are in trouble. You can reach him at marcot@phparch.com.

TIPS & TRICKS

Tips & Tricks

By John W. Holmes

Extended Sessions

Sessions don't have to end when the user closes their browser. You can adjust the setting of session.cookie_lifetime in php.ini to give a lifetime, in seconds, of the session cookie. If you do so, the session cookie will persist, even if the user closes the browser before returning back to your site. If you adjust this setting, you should also adjust the session.gc_maxlifetime setting, also. This setting, which controls when the garbage cleanup will delete "stale" data from the session files. If you have the session cookies persisting for a longer amount of time, then the actual data in the session files should persist for that amount of time before they are cleaned up, alsoas well. If you are not using the default session handler, then the garbage collection does not occur for your sessions and you must handle this yourself.

Now, whether there would ever be a need to do this is another question. Extending the life of the session means there are going to be more active sessions at a time on your server and more files in the session folder. Whether this is an issue or not depends on the amount of traffic on your server. Having that same session_id persist will also give malicious users more time to hijack a session. Things like this should be taken into consideration before you decide to extend the life of your session.

Speaking of Sessions

ACROS Security (www.acros.si) published a paper in December entitled "Session Fixation Vulnerability in Web-based Applications." The paper discusses how a hacker can "issue" a session ID to a user as they log onto your web application. PHP is described as having a "permissive" session management mechanism, because it will take any value of PHPSESSID and create the session with that name. That means loading a page that starts a session with ?PHPSESSID=1234 will create a session with ID=1234 and a "php_1234" file in the sessions directory.

Now if a malicious user can get another user to click on a link to your application that defines a session ID (and we all know how eager most people are to click on links), they no longer have to intercept the session ID, they already know what it is. They can wait for the user to log in and then simply load up a protected page and also pass along the same session ID they just forced your application to use. Now they are basically logged in as the user and have access to their pages and data. Yes, it is a little more complicated; the malicious user would have to know when the user has logged on, what page to request, etc. You can also implement checks of the IP address or referrer, but since those can be set by the malicious user or can even change for valid users, you're not going to stop someone who is dedicated.

So how do you stop them? Well, the valid user still has to log on at some point and that's when you make sure they are assigned a unique session ID that didn't come from outside of your application. On the login page, use the following code :

```
session_id(md5(uniqid(rand(),1)));
session_start();
```

To create a unique session ID for the user and ignore anything that might have been passed by a malicious user. This can only be done on the login page, otherwise you'll be creating new unique session IDs with every request. After the user logs in and throughout the rest of your site, you must now trust that the session ID passed is unique and has been created by your program. Malicious users will now be reduced to intercepting or guessing valid session IDs to get into your application.

You can even make PHP have a "strict" session management system by registering the session IDs you create in a database or file and ensure that subsequent requests match one of those session IDs that your application created.

Less phpinfo()

The `phpinfo()` function is extremely useful for checking to see if you have installed PHP correctly and for verifying a lot of the internal settings of PHP. It can also be useful in debugging scripts as you write them, as it can print out the values of the EGPCS (Environment, Get, Post, Cookie and Session) variables. To make this easier, the function will take an argument that lets you pick which section of the page you want printed. Figure 2 shows the different values you can pass and what part of `phpinfo()` will be shown with each value. Rather than printing out the whole page, you can use phpinfo(32) in the "debugging mode" of your program to keep an eye on what's being passed to your scripts.

Manage Textbox Lengths

Tim Cowan submitted a tip that can help you manage the length of your textboxes in forms. The problem is that if you don't define a `MAXLENGTH` for your textbox, then the user can type 50 or more characters into it, not knowing that your database is going to truncate it at 20. Of course, you can always set the `MAXLENGTH` to 20 as you create your code, but then when you change your database table, you have to go back and edit all of the textboxes to reflect the new text length.

So the suggestion is to create a function that'll read your table and look for columns that have a length. These lengths can then be returned to your script to be used in you <input> form elements. Figure 3 shows an example of two functions that will look for the lengths of VARCHAR and CHAR columns. The function returns an array with the name of the table column as the key and the length of the column as the value.

If you have a column 'Name' in a table 'test' that is defined as a VARCHAR(25), the functions will return an array such as the one shown below.

```
Array ( [Name] => 25 );
```

You can then use that value when creating your form elements. For example, say the results shown above were assigned to the variable $a. You would use it like the following as shown here:

```
<input type="Name' maxlength="<?=$a['Name']?>">
```

Of course, you should adapt the functions to your

Figure 2

Name (constant)	Value	Description
INFO_GENERAL	1	The configuration line, php.ini location, build date, Web Server, System and more.
INFO_CREDITS	2	PHP 4 Credits. See also phpcredits().
INFO_CONFIGURATION	4	Current Local and Master values for php directives. See also ini_get().
INFO_MODULES	8	Loaded modules and their respective settings.
INFO_ENVIRONMENT	16	Environment Variable information that's also available in $_ENV.
INFO_VARIABLES	32	Shows all predefined variables from EGPCS (Environment, GET, POST, Cookie, Server).
INFO_LICENSE	64	PHP License information. See also the license faq.
INFO_ALL	-1	Shows all of the above. This is the default value.

Figure 3

```
function get_lengths($table)
{
    $retval = FALSE;

    $result = mysql_query("SHOW CREATE TABLE $table");
    if($result)
    {
        $create = mysql_result($result,0,1);

        preg_match_all("/`([a-z0-9_]+)`\s(var)?char\(([0-9]+)\)/i",$create,$matches);

        $cnt = count($matches[1]);

        for($x=0;$x<$cnt;$x++)
        { $retval[$matches[1][$x]] = $matches[3][$x]; }
    }

    return $retval;
}

function get_lengths2($table)
{
    $retval = FALSE;

    $result = mysql_query("DESC $table");
    if($row = mysql_fetch_assoc($result))
    {
        do
        {
            if(preg_match("/char\(([0-9]+)\)/i",$row['Type'],$match))
            { $retval[$row['Field']] = $match[1]; }
        }while($row = mysql_fetch_assoc($result));
    }

    return $retval;
}
```

own needs. The ones shown are only examples. They both run at about the same speed during testing. You could also look for INT or date/time columns. Also, remember that this is a simple client side check that can be bypassed. You should properly validate all of the values on the server side before you do anything with them.

Thanks to Tim for this tip.

PHP 4.3 and SSL

Since PHP 4.3 is officially out, there are a couple new features relating to SSL that should be mentioned.

If you have URL wrappers enabled in php.ini and OpenSSL installed, you can use `fopen()` to open an HTTPS connection to a web server. This can be used, for example, to securely post information to a form on another web site by passing information through the URL.

```
fopen("https://domain/script.php?var=1&var=2',
      'r');
```

Then script.php can respond based on the passed values. This would be an alternative to CURL or using external programs for simple cases such as this.

You can now connect to MySQL over SSL, as well. A fifth parameter to `mysql_connect()` was added to pass 'client_flags'. Setting this fifth parameter to `MYSQL_CLIENT_SSL` will cause the connection to be made over SSL. This is going to be ideal for applications that have the web server on one computer and the database on another. This will ensure the data being transmitted between the two systems in protected.

Submit Your Own Tips & Tricks

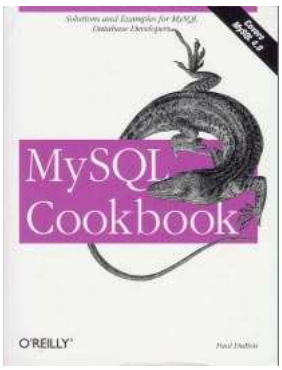
If you have any suggestions or fancy tricks that you use, be sure to submit them to us at tipsntricks@phparch.com. We'll give away a free one-year subscription to php|a to the authors of all the tips we publish!

php|a

John Holmes is a First Lieutenant in the U.S. Army and a freelance PHP and MySQL programmer. He has been programming in PHP for over 4 years and loves every minute of it. He is currently serving at Ft. Gordon, Georgia as a Company Commander with his wife, son, and another one on the way.

For Your Reading Pleasure

MySQL Cookbook
by Paul DuBois
Published by O'Reilly
$49.95 (USA)
$77.95 (Canada)



Raise your hand if you've ever heard someone say that MySQL is a "simple" database management system that shouldn't be used in complex enterprise-level applications because of its lack of functionality.

We can't see you from here, but we're sure that quite a few hands are now up in the air and, if you're like us, this is something that you can only find very frustating.

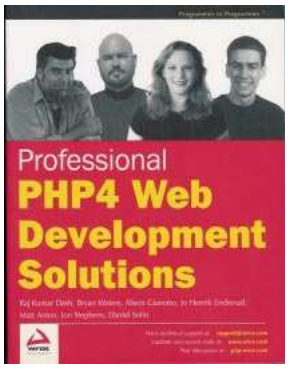
Receiving a book like MySQL Cookbook for review is, therefore, a great pleasure; it's not everyday that one can get his hands on a 1000-page weapon against the MySQL detractors!

The Cookbook covers in almost excrutiating detail all the aspects of MySQL—from using its text-based client to statistical techniques and transaction management. Paul DuBois does an excellent job at dissecting each topic from a number of different perspectives. For example, when covering the MySQL client tool, he starts from the basic "how-to-run-your-query" tutorial and then goes deep into the details of generating neatly-formatted HTML and XML output. Similarly, when discussing transactions, the book deals with their implementation in several languages—including PHP—on top of the pure SQL approach.

The section on statistical analysis, in particular, is very rich in valuable techniques for taking advantage of MySQL's engine in ways that are not immediately obvious, such as combining grouping and aggregators to produce statistics on partial groups.

While this book is mostly SQL-centric (as it should be), whenever the author introduces a web-based example, he does so in a variety of languages—usually Perl, PHP and Python, thus giving his readers a great opportunity not only to learn more about MySQL, but also to compare the various degrees of complexity in dealing with it through a particular scripting platform.

PHP4 Web Development Solutions
by K. Dash, B.Waters, A. Gianotto, J.H. Endrerud et al.
Published by Wrox Press
$49.99 (USA)
$77.99 (Canada)



If you expect this book to teach you PHP, I hope you kept the purchase receipt. It is not for the beginner. More like someone who has been working with PHP and wants quick solutions for common problems. Here's a quote from the introduction:

"This book targets experienced PHP Programmers who either have some PHP experience, or the wherewithal to learn rapidly via the case studies."

This book is a compilation of tutorials for configuring, and implementing different web applications. Each tutorial has an introduction which includes the purpose behind the solution and the practical uses behind it. There are quite a few applications geared towards WAP and free tools from the Internet to make the job easier (ie. Smarty template engine, PEAR::DB to name two) . Also, there is extensive use of Classes in all examples (Can you smell that? Smells like PHP5) .

There is an interesting project to build a web corpus (indexing unique words used in a web content for a search engine). The basics of building a search engine are here. And, of course, what good is a PHP solutions book without a decent example of a Content Management System?

Some projects use a fictional backdrop, and where to apply the application is left as an (offline) exercise for the reader. For example, the Mission Control Wireless Job Board has an interesting "SPY" story, where clients assign agents to "Missions". Once a client chooses an agent, the agent receives a text message on their wireless device. Then, the agent can use their wireless device to contact the client and get all the details regarding the "Mission". Now, where have we seen this before? "Mission" something...well, this chapter makes it look more possible than ever. There is strong emphasis on WML.

There are some inconsistencies when it comes to presenting the material. For example, not all projects have a flowchart where the user's experience is shown (a logical DFD). It's a very useful concept for programmers where logic is everything. A flowchart is very visual and helpful to understand how all the pages work together.

There are no fancy call-outs and reminder 'icons' in the book. It gets right to the point. Now this might not be a very friendly way to keep your readers interested, but it's fine for the experienced programmer. Although the presentation is very plain, it does give you what you need to know about the code. Like most well written technical books, all the code used in the book is available from WROX. It's assumed the reader downloads this from the website. It would have been more convenient to have everything on a CD, but I guess having it on a website makes it possible to update/patch any code necessary.

Speaking of code, it is well commented -- sometimes. Let me elaborate: the code in the Classified Ads Board is very well commented, while the code in the Paranormal News Service doesn't have a SINGLE line of comment. I found that very odd.

At the end of every chapter, there is always a list of improvements that can be made to the project. To save paper, the details are posted on the WROX website (http://www.wrox.com). Some of the case studies do build on each other. The

Content Management one is a good example. Moreover, as the book progresses, every case study gets a little more complex.

In a nutshell, it's a white paper for a lot of case studies found on the Wrox website. The experienced programmer might just download all the code and not buy the book, but that won't work. You see, there are some critical pieces of information in the book. Without those, it would be like looking at the source code for a DLL written in VB.

Let's Call It The Unknown Language

By Marco Tabini

The release of PHP 4.3 marks a significant step forward in the development of our beloved language. In fact, it has been picked up by a number of open-source news outlets, such as NewsForge (http://www.newsforge.com) and SlashDot (http://www.slashdot.org). This latter site, in particular, is often home to some rather colourful discussions—thankfully, there are lots of opinionated people in the OSS world—and the 4.3 news release was no exception. Although the various threads sometimes covered useful topics—such as the new features offered by the latest release—lots of people promptly jumped on board to turn the posting into yet another "my-language-is-better-than-yours" Holy War of Computer Developers.

Why people even engage in such discussions is beyond me, and I usually retreat in fear when I see one. However, this particular discussion did raise one important point: many—too many—people out there have no idea what PHP can really do. Most developers who tried it in its early incarnations (most likely version 3) think of it as a "nice hacking tool" for putting web pages together when you're in a hurry. They are completely unaware of the impressive steps forward that PHP has taken since then.

The temptation to react to this reality by shaking of our heads and muttering something like "too bad for them" is strong (at least, it is for me). That's a very dangerous attitude, however, for the responsibility to ensure that the world at large understands how advanced PHP has become as a serious development tool falls on the shoulders of the PHP community in general. Widespread adoption and understanding is the only way to ensure a high level of support for the technologies that we use everyday.

The PHP Group can't do this by itself. For one thing, it isn't geared up for this type of task, as it is mainly a technical team that has its collective hands full with the development of the language. In this month's interview on the pages of this magazine, Zend's President and CEO Doron Gerstel lends creedence to this idea: it's necessary for PHP users and commercial vendors to step up to the task and devote at least some of their resources to advocacy and publicity for PHP in general.

We at php|a try to do our part—after all, a magazine dedicated to PHP is almost purely a tool of advocacy—but we recognize that there is still a lot of road to travel, and hope to fulfill our role even better with some initiatives that we are planning for the months ahead. I, for one, can only hope that the rest of the PHP community steps up to the plate and that together we can build a better business environment for PHP.